A Framework for Adaptive Routing in Multicomputer Networks

John Y. Ngai

Computer Science Department
California Institute of Technology

Caltech-CS-TR-89-09

A Framework for Adaptive Routing in Multicomputer Networks

Thesis by

John Y. Ngai

In Partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy

California Institute of Technology
Pasadena, California

1989

(Submitted May 17, 1989)

Caltech–CS–TR–89–09

Acknowledgments

During the years that I stayed as a student at Caltech, numerous people either taught me or helped me along the way. A detailed list of everyone is simply much too long to be included here; however, a number of individuals definitely stand out and deserve special mention. First, I would like to express my deepest thanks to my thesis adviser, Chuck Seitz, for his support and encouragement during all these years. This thesis simply would not have been possible without his guidance and inspiration. I thank all the members of my reading committee: K. Mani Chandy, Joel Franklin, Alain Martin, Edward Posner and Chuck Seitz for their helpful suggestions and constructive criticisms.

I would like to express special thanks to Arlene DesJardins, who has made this department a pleasant place in which to study and to do research. She has always been responsive to my requests for help, and made sure that things were always there when I needed them. Special thanks also goes to Dian De Sha, who spent weeks sieving through my idiosyncratic and sometimes baroque English, and made numerous suggestions that greatly improved the readability of the final manuscript. Any remaining bugs are obviously my sole responsibility.

Wen-King Su, Jakov Seizović, Don Speck, Michael Lichter, and Joe Beckenbach all deserve special thanks for providing an excellent computing environment for me to carry out my work. They have put up with my numerous questions and demands along the way, and have never hesitated to help whenever I asked.

I like to thank Nancy Zachariasen, Angela Blackwell, and Cindy Ferrini for their help in locating reference books and research materials, and I am grateful to Bill Athas, Charles Flaig, Craig Steele, and Wen-King Su for their helpful discussions. The friendship of Marcel van der Goot, Tony Lee, Dražen Borković, Nanette Boden, and many others that has made Caltech an enjoyable place to be is also appreciated.

Finally, deep thanks go to my parents, who brought me up in a warm and happy family, for their love which is beyond words, and for their resolute determination to provide me with the best opportunity in education. I also owe special thanks to my brother, Peter Ngai, for his constant encouragement. Above all, I thank God for His infinite grace and love.

The research described in this thesis was sponsored in part by the Defense Advanced Research Projects Agency, DARPA Order number 6202, and monitored by the Office of Naval Research under contract number N00014-87-K-0745; and in part by grants from Intel Scientific Computers and Symult Computer Research Division.

Abstract

Message-passing concurrent computers, also known as multicomputers, such as the Caltech Cosmic Cube [47] and its commercial descendents, consist of many computing nodes that interact with each other by sending and receiving messages over communication channels between the nodes. The communication networks of the second-generation machines, such as the Symult Series 2010 and the Intel iPSC2 [2], employ an oblivious wormhole-routing technique that guarantees deadlock freedom. The network performance of this highly evolved oblivious technique has reached a limit of being capable of delivering, under random traffic, a stable maximum sustained throughput of $\approx$ 45 to 50% of the limit set by the network bisection bandwidth, while maintaining acceptable network latency. This thesis examines the possibility of performing adaptive routing as an approach to further improving upon the performance and reliability of these networks. In an adaptive multipath routing scheme, message trajectories are no longer deterministic, but are continuously perturbed by local message loading. Message packets will tend to follow their shortest-distance routes to destinations in normal traffic loading, but can be detoured to longer but less-loaded routes as local congestion occurs.

A simple adaptive cut-through packet-switching framework is described, and a number of fundamental issues concerning the theoretical feasibility of the adaptive approach are studied. Freedom of communication deadlock is achieved by following a coherent channel protocol and by applying voluntary misrouting as needed. Packet deliveries are assured by resolving channel-access conflicts according to a priority assignment. Fairness of network access is assured either by sending round-trip packets or by having each node follow a local injection-synchronization protocol.

The performance behavior of the proposed adaptive cut-through framework is studied with stochastic modeling and analysis, as well as through extensive simulation experiments for the 2D and 3D rectilinear networks. Theoretical bounds on various average network-performance metrics are derived for these rectilinear networks. These bounds provide a standard frame of reference for interpreting the performance results.

In addition to the potential gain in network performance, the adaptive approach offers the potential for exploiting the inherent path redundancy found in richly connected networks in order to perform fault-tolerant routing. Two convexity-related notions are introduced to characterize the conditions under which our adaptive routing formulation is adequate to provide fault-tolerant routing, with minimal change in routing hardware. The effectiveness of these notions is studied through extensive simulations. The 2D octagonal-mesh network is suggested; this displays excellent fault-tolerant potential under the adaptive routing framework. Both performance and reliability behaviors of the octagonal mesh are studied in detail.

A number of implementation issues are examined. Encoding schemes for packet headers that admit simple incremental updates while providing all necessary routing information in the first flit of a relatively narrow flit width are developed. A pipelined control structure that allows a packet to cut through an intermediate node with a minimum delay of two cycles is described. A distributed clocking scheme is developed that eliminates the problem of global clock-signal distribution. Under this clocking scheme, the adaptive routers can be tessellated to form a network of arbitrary size.

Contents

List of Figures

Chapter 1

Introduction

The advances in VLSI technology allow us to build large-scale computing structures consisting of many processing elements that operate concurrently. Among these, message-passing concurrent computers, also known as ***multicomputers***, such as the Caltech Cosmic Cube [47] and its commercial descendents [2] consist of many computing nodes that interact with each other by sending and receiving messages over communication channels between the nodes (see Figure 1.1). The communication networks of the medium-grain second-generation machines, such as the Symult Series 2010 and the Intel iPSC/2, employ an ***oblivious*** wormhole routing technique that guarantees communication-deadlock freedom. For fine-grain message-passing concurrent machines, such as the Caltech Mosaic [32], it has become progressively more difficult to achieve the desired network performance necessary to support fine-grain concurrent computations [3]. In particular, the bisection-bandwidth capacity of physically realizable communication networks grows at a rate that is slower than the expected message traffic, which is likely to be at least linear in the total number of computing nodes [13].

To improve the overall performance of these machines, it is necessary to improve the performance of both the computing nodes and the underlying communication networks. Continuing advances in instruction-interpreting processors [3,13,32] promise to deliver more raw power to the individual computing nodes in these machines. On the other hand, the network performance of this highly evolved oblivious technique has reached the limit of delivering, under random traffic, a ***stable*** maximum sustained throughput of ≈ 45 to 50% of the limit set by the network-bisection bandwidth while maintaining acceptable network latency. Further improvements on these networks will require an adaptive utilization of available network bandwidth to help diffuse local congestion. In an adaptive multipath routing scheme, message trajectories are no longer unique and deterministic, but are continuously perturbed by local message loading. Message packets will tend to follow their own shortest-distance routes to destinations in normal traffic loading, but can be detoured to other longer but less-loaded routes as local congestion occurs. Furthermore, an adaptive routing scheme can potentially support a hot-spot throughput that is equal to the total communication bandwidth of the generating node rather than to the bandwidth of a single communication channel [13], as it is in the case of an oblivious scheme.

In addition to the potential gain in network performance, an adaptive multipath routing scheme also opens up the possibility of enhancing the reliability of these large-

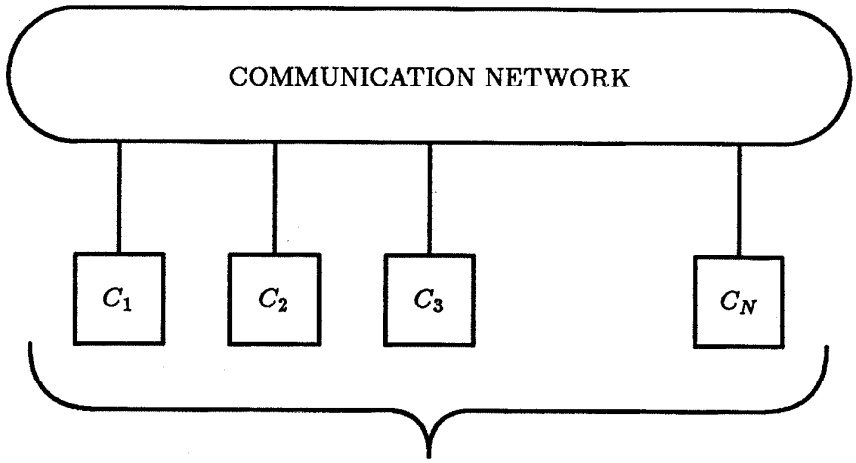

Figure 1.1: Programmer's Model of a Multicomputer

scale networks by performing *fault-tolerant* routing. In particular, we observe that the communication networks that are popular among existing multicomputers are already very richly connected. An adaptive multipath routing scheme has the potential for taking advantage of the inherent *path redundancy* in these richly-connected networks; this is otherwise impossible to exploit under the oblivious restriction.

1.1 Multicomputer Networks

While the specific details of multicomputer communication networks differ depending on the connection topologies and the adopted routing schemes, it is still possible to discern a number of common operating characteristics among the high-performance members. In this section, we shall review these general operating characteristics for multicomputer networks, and shall compare them with those found in the more familiar geographically distributed networks. To start, we observe that one of the common characteristics shared among multicomputers in general, and the fine-grain machines in particular, is that the hardware resource per node is rather limited. For example, given the current VLSI technology as the implementation medium, we can generally expect the average medium-grain machines to have megabytes of main memory, whereas the corresponding average in a fine-grain machine is likely to remain at about a few tens of kilobytes. An immediate consequence of having such small memory capability per node is that the amount of internal message buffer that each node can afford to devote to supporting routing will be very limited.

These characteristics are in contrast to those found in loosely-coupled networks, such as the ARPANET [39]. For example, in these loosely-coupled networks, message buffering space per node is usually much more abundant. In light of the continuing decline in storage cost, any shortage experienced at a node of these networks should be considered as an underdesign. Communication deadlock in such networks is practically nonexistent; hence, deadlock detection and recovery schemes can be used to handle the remaining rare occasions without compromising the performance of these networks. Such schemes

can hardly be justified in tightly-coupled multicomputer networks intended to support massively concurrent computations, and the issue of guaranteeing deadlock freedom must lie at the heart of any practical routing scheme proposed for such networks.

Another common characteristic generally shared among multicomputer networks designed to support massive concurrency is that, for performance reasons, these networks are all *physically* very tightly coupled, where the multicomputer nodes are physically clustered together in the same spatial locale. As a result, when driven properly, the signal-propagation delay across a communication channel need not be any slower than the on-chip circuit delay. This small signal-propagation delay creates an opportunity that is absent in the loosely-coupled networks. Specifically, it affords the possibility of a cycle-by-cycle flow-control mechanism, which has been employed successfully in the oblivious wormhole routing implementations [11,14]. In fact, it allows us to regulate the incoming and outgoing message traffic rate on the same cycle-by-cycle basis. As we shall see, this makes it possible to implement a *misrouting* discipline that underscores our freedom from communication-deadlock result. Implementing a similar scheme in a loosely-coupled network would be prohibitively inefficient, if not technologically infeasible.

In much the same way, the multicomputer networks, unlike geographically distributed networks, are usually installed in well-protected environments; hence, the signal transmission error rate across a channel is extremely small and generally negligible. In fact, there is no reason to expect the transmission error rate to be any worse than that of a memory fetch inside a processor system. This is in contrast to the loosely-coupled *store-and-forward* routing networks that cover wide geographical areas, and where the communication links could be subjected to adverse external environmental conditions. The extremely low transmission error rate attained in the tightly-coupled multicomputer networks creates the opportunity to allow us to employ more aggressive routing techniques, such as *virtual cut-through* switching [25] to enhance the performance of these networks, and to reduce the requirements for internal buffering. An overall *end-to-end* error-detection and -correction scheme would be sufficient to handle the few rare error occurrences in these networks, and would eliminate the need for the *point-to-point* error detection and correction that necessitates the store-and-forward mode of message routing. Cut-through switching will be discussed in greater detail in the next section.

Perhaps the main impact of being so physically tightly-coupled is that the multicomputer networks operate typically at two orders of magnitude faster than the local area networks, such as the Ethernet, or the geographically distributed networks, such as those based on T1 lines. For example, the Ethernet runs at 10 Mb/s and the T1 carrier runs at 1.544 Mb/s, whereas under the current routing chip technology, the multicomputer channel speed is approaching 1 Gb/s [2,14]. The immediate result of having such fast channels is that we typically have only 10's of nanoseconds to make the routing decisions in cut-through switching, rather than having microseconds or even milliseconds as it is in the case of the slower networks. At such high switching speed, the amount of information that a routing algorithm can afford to examine during decision making is severely constrained. In any case, the routing protocols employed in multicomputer networks must remain very simple.

Yet another common characteristic generally shared among multicomputer networks is that these networks almost always employ very *regular* network topologies for their

connections. One immediate advantage of using regular connection topologies is that they usually allow one to define simple algorithmic routing procedures that eliminate the requirements to store and to consult routing tables. This is desirable in light of the very limited available resource at each node, and is also absolutely essential in these high speed multicomputer networks where the routing function must be implemented directly in hardware. Another advantage of using regular topologies is that they also suggest systematic layouts, and this is convenient for configuring arbitrarily extensible systems. The elimination of routing tables, however, raises the new issue of how to perform *fault-tolerant* message routing when some of the nodes and channels are broken, thus destroying the regularity of the original network connection topology.

1.2 Cut-Through Switching

It is clear that in order for any routing scheme to compete favorably with the existing highly evolved oblivious scheme, it must employ a switching technique akin to virtual cut-through. The main advantage of cut-through switching or its blocking variant known as *wormhole* routing [11], which is used in the existing oblivious routers, is the very low message latency attained under light-traffic conditions. In cut-through routing, packets are forwarded to their next intermediate nodes along the route as soon as sufficient header information has arrived to allow its outgoing channel to be selected. This is different from the conventional store-and-forward networks, where entire packets must be received before being forwarded. Figure 1.2 depicts a graphic comparison of the total packet latencies of the two approaches: In store-and-forward routing, the total latency is the *product* of the length of the packet and the number of hops the packet has to travel. In cut-through routing, the total latency becomes instead the *sum* of the two quantities. For almost any reasonable message length, this will result in a substantial reduction of the total message latency. Not only is the latency reduced substantially for messages that traverse more than one channel; for messages with lengths that are long compared to the message distances, it also becomes relatively *insensitive* to the distance. This insensitivity in message distances reduces the importance of message locality in concurrent computations. It allows nonlocal communication to be used without incurring much degradation in message latency in an environment that operates under moderate traffic density [13].

We distinguish between wormhole routing, which is employed in the oblivious routers used in such existing second-generation multicomputers as the Symult Series 2010 and the Intel iPSC/2, and the cut-through routing technique, which is adopted in this thesis. These two techniques have been motivated by different conceptual models. In wormhole routing, the head of a packet will be immediately forwarded along its route whenever there is no conflict in channel access, or when the channel becomes idle. When a channel-access conflict occurs, the packet is blocked behind the busy channel, waiting for it to become available. The body of the packet occupies the channels along its route, whereas the tail of the packet releases these occupied channels as it makes its way toward the destination. In cut-through routing, packets behave exactly as they do in the wormhole technique, as long as no channel-access conflict occurs. When the requested channels are busy, however, the entire packet will be *stored* in the intermediate node at the collision spot. Because of the requirement to store packets at intermediate nodes, long messages

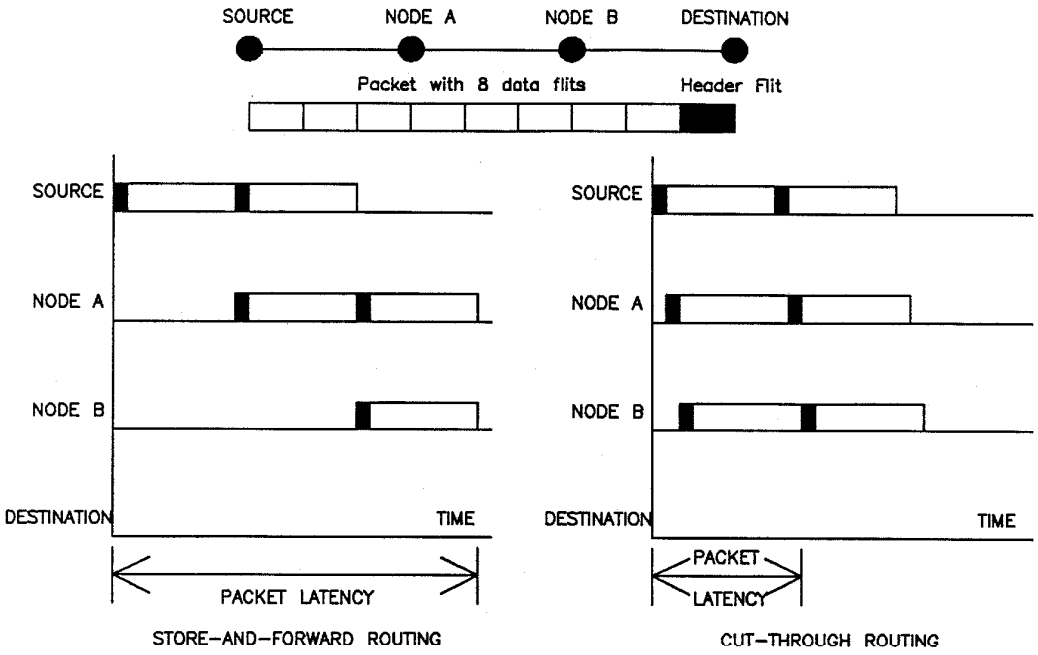

Figure 1.2: Store-and-Forward *versus* Cut-Through Routing

must be partitioned into shorter packets for transmission.

The cut-through routing technique can be considered as a hybrid between circuit switching and store-and-forward switching. In particular, under relatively light traffic density, channel-access conflicts will be rare, and, therefore, the network will operate predominantly in circuit-switching mode. As the traffic density increases, collisions will become increasingly common, until heavy traffic density causes the network to operate predominantly in the store-and-forward switching mode. The operation of wormhole routing, on the other hand, can be considered as an improvement over conventional circuit switching in that both the circuit *setup* and the circuit *release* operations occur concurrently with the actual transfer of message data. Because wormhole routing is primarily a circuit-switching technique, the only reason to partition long messages into shorter packets for transmission is to achieve stronger fairness in channel usage. Furthermore, message arrival order between each source-destination pair is always automatically preserved.

1.3 Adaptive Multipath Routing

In order to establish the background for the presentation of the adaptive routing framework in the next chapter, in this section, we introduce informally the basic ideas and considerations behind the adaptive ***multipath*** routing approach studied in this thesis. We shall illustrate the ideas using a 3D rectilinear mesh network. These considerations are also directly applicable to other regular communication structures.

A routing algorithm is *oblivious* [8,12] if the route followed by a message is predetermined by its source–destination pair. For rectilinear meshes, under the consumption assumption, *ie*, any message arriving at its destination will be consumed, a very simple oblivious scheme that guarantees communication-deadlock freedom is to route the messages in dimension order. For example, on a 3D mesh, a possible scheme is always to route first along the X-dimension, reducing ΔX to zero, and then along the Y-dimension, reducing ΔY to zero, and finally along the Z-dimension, reducing ΔZ to zero [14]. To see that this scheme is indeed deadlock-free, observe that because of the dimension ordering, a message waiting for a busy $+Z$ direction channel must be destined to a node that lies along a Z-axis having the same X- and Y-coordinates, and with a Z-coordinate that is larger than the Z-coordinate at the place where blocking occurs. Furthermore, neither this message nor the one blocking it and so forth, will request any channel other than the $+Z$ direction channels. This reduces the dependencies into a linear list. Given the consumption assumption, we can guarantee that any message routed along the Z-dimension will eventually be delivered. The same argument can now be applied to the messages blocked along Y-dimension channels. They must be blocked either by packets destined to nodes along the same Y-axis, or by packets that are waiting for the Z-dimension channels. In both cases, the blocking will eventually be released. Therefore, all messages routed along the Y-dimension channels will eventually be delivered. A similar argument applied to X-dimension channels establishes delivery for all messages in the network.

On the other hand, we observe that on the 3D mesh, if we want to forward a message to a node that is ΔX hops away along the X-dimension, ΔY hops away along the Y-dimension, and ΔZ hops away along the Z-dimension, then there are a total of $\frac{(\Delta X+\Delta Y+\Delta Z)!}{\Delta X!\Delta Y!\Delta Z!}$ different shortest-distance routes joining the message to its destination. At places where more than one dimension has a nonzero delta quantity, sending the message out along either one of these dimensions toward its destination may be considered equally *profitable*, in that each will bring the message one hop closer to its destination. Indeed, depending on a message's current location relative to its destination, it may be profitably routed out in one of a total of $3^3-1 = 26$ different possible output-channel combinations. An adaptive control can take advantage of the extra flexibility that some messages have of being able to route profitably in more than one direction. This is different from the oblivious routing strategy, where the entire message route is determined *solely* by its source and destination nodes.

For example, consider the situation depicted in Figure 1.3, where we have a node with two incoming messages and two messages that are cutting through the node. The two incoming messages request, respectively, the output channels $+Y$ or $+Z$, and $-Y$ or $-Z$. If we follow the oblivious deadlock-free strategy of always routing along the Y dimension before the Z dimension, then both messages will be blocked in front of the Y-dimension channels. Under an adaptive control, both messages can immediately cut through the Z-dimension channels. However, this same example also exposes a serious communication-deadlock problem in multipath routing that demands a solution. In particular, since the two incoming messages are forwarded across the Z-dimension channels before they have finished traversing their Y-dimension channels, we are no longer following the dimension-order routing restriction. This can create cyclic dependencies among the messages and cause communication deadlock. For example, in wormhole routing, deadlock can now

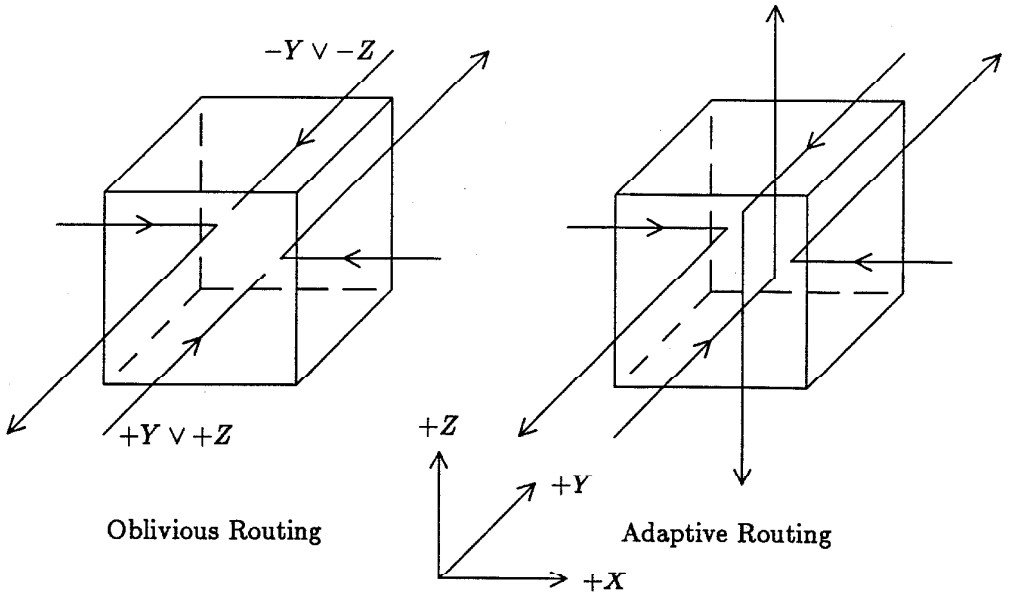

Figure 1.3: A Simple Adaptive Routing Example

occur when a message becomes a member of a set of messages blocked in a cycle where each is unable to proceed until the next one releases its occupied channel.

Methods to prevent communication deadlock have been intensively researched and many schemes exist; of these, the methods of structured buffer pools [37] and virtual channels [12] are representative. In essence, all of these methods approach the problem by remapping any dependency that is potentially cyclic into a corresponding acyclic dependency structure. However, these methods require either a buffer size per node that grows with the diameter of the network, or a large number of virtual channels per node in order to allow multipath routing. In this thesis, we adopt a different and very simple technique that is independent of network size and topology, by using ***misrouting***. Misrouting was first suggested in [8] in a different context for networks that employ data-exchange operations. In our approach, a misrouting discipline is adopted to eliminate communication deadlock. It allows us to follow a no-blocking convention that is equivalent to breaking the potentially cyclic dependencies into disjoint paths of unit length. However, this elimination of communication deadlock is obtained with a cost. In allowing misrouting, we have created the burden to demonstrate network progress in the form of assuring message delivery. This and other progress-related issues will be examined in detail in Chapter 2.

A fundamental characteristic of our adaptive routing control is that it forwards any message passing through the node according to a decision policy based entirely on ***locally*** available information. In particular, we depend on the ability of the adaptive multipath routing control to direct messages along many alternate routes in order to smooth out any temporary congestion and disperse the local traffic hot spots. The more numerous the number of alternate routes a message can follow, the more effective this multipath strategy will be. ***Adaptation*** occurs entirely at the local level where messages

are forwarded through one of their profitable channels depending on the specific traffic engagement at the time of decision.

1.4 Overview of Thesis

We now give a very brief overview of this thesis. In Chapter 2, we present the adaptive cut-through packet switching framework adopted in this thesis. This framework provides the context for the discussion of several fundamental issues concerning the theoretical *feasibility* of our adaptive routing approach. In particular, the discussion focuses on the issues of assuring freedom from communication deadlock, assuring progress inside the network, *ie*, the eventual delivery of every message packet in transit in the network; and assuring fairness in network access, *ie*, the eventual injection of any queued packet into the network. Communication deadlock is rendered a non-issue by the adoption of a misrouting discipline. Packet delivery is guaranteed by choosing a suitable priority assignment for each packet. Packet injection is guaranteed by introducing a cooperative injection protocol between neighboring nodes. The successful resolution of these issues is important for any message communication network supporting reliable concurrent computation.

In Chapter 3, we study the performance of the proposed adaptive cut-through routing scheme from stochastic modeling and analysis, as well as through extensive simulation experiments. In particular, our study focuses on the two-dimensional and three-dimensional rectilinear meshes and tori. These low-dimensional rectilinear networks represent connection topologies that are realistically implementable, are regular, and provide simple algorithmic routing procedures that enable direct hardware realization of the routing control. We extend the packet-injection-assurance protocol described in Chapter 2 to provide more spontaneous congestion control for the adaptive schemes. We study the effectiveness of this congestion-control protocol and confirm it through simulations. In this chapter, we also present theoretical bounds on the various average network performance metrics for random traffic over these networks. These bounds provide a standard frame of reference to interpret the performance results.

In Chapter 4, we investigate the potential network reliability enchancement offered by the adaptive approach through exploiting the inherent path redundancy in the richly connected networks to perform fault-tolerant routing. The main focus of the study is how to accommodate the increasing irregularity in a network with faults, because irregularity renders algorithmic routing difficult and complicated. We introduce two convexity-related notions that characterize the conditions under which our adaptive routing formulation is adequate for providing fault-tolerant routing in the survived networks, and with little change in the original hardware. We study the effectiveness of these notions through extensive simulations. Based on the simulation results, we suggest the 2D octagonally-connected mesh network, which displays excellent fault-tolerant potential under the adaptive routing framework. We study both the performance and the reliability behaviors of the octagonal mesh.

In Chapter 5, we examine several major implementation issues. In particular, we argue for the use of the congestion-control protocol studied in Chapter 3 as a much simpler practical alternative to the use of packet priorities in enhancing network progress. We describe a number of header encoding schemes for these rectilinear and octagonal meshes

that admit simple incremental update while providing all necessary routing information in the first flit with a relatively narrow flit width; a possible buffer organization capable of storing variable-length packets; and a pipelined organization for implementing the adaptive routing control. We suggest an iterative network realization of the assignment decision process, and describe a simple distributed clocking scheme that eliminates the problem of global clock signal distribution. Under this clocking scheme, the adaptive routers can be tessellated to form a network of arbitrary size.

In Chapter 6, we conclude this thesis with the discussion of a number of issues and open problems that have been suggested during the course of this investigation.

Chapter 2

Feasibility

If there is any advantage to be gained by making adaptive routing decisions rather than oblivious ones, it must come from the flexibility to exploit alternative routes. Such flexibility must be introduced in ways that still guarantee freedom from deadlock. The concept of *virtual channels* [12] provides a way to develop deadlock-free routing algorithms in arbitrary direct networks that employ the wormhole routing style. While it is possible in certain special cases to allow for multiple alternative routes using virtual channels, the task becomes progressively more difficult in more complex routing schemes and network topologies. The concept of a *structured buffer pool* [37] developed for store-and-forward computer communication networks provides a general approach for introducing alternate deadlock-free routes into an arbitrary packet switching network. However, these schemes in general require a buffer pool that grows with the network size, and hence are not feasible in fine-grain multicomputer communication networks, where resources at the node level are finite and scarce, and the network is arbitrarily extensible.

In this thesis, we consider networks that employ a *coherent transfer* [42] protocol, rather than the *block until clear then forward* mechanism used in wormhole [11] routing or the *packet acknowledge* mechanism used in *store-and-forward* [6] routing, as the basic node-to-node data transfer protocol. By enforcing the rule that all communicating parties agree to maintain the local invariance of always being able to accept another round of requests from neighbors through preemption of the requested resources if necessary, the coherent transfer model avoids the necessity of having to *block*, *ie*, hold a packet and wait until the requested resource becomes available and has been acknowledged. Data transfer is guaranteed locally between every pair of adjacent nodes that act as partners in the data transfer operations. Adopting voluntary misrouting as a strategy to overcome the constraint of fixed finite local resources, however, creates the need to demonstrate message delivery and network progress.

In section 2.1 of this chapter, we present a simple model for the discussion of adaptive routing schemes that allows us to examine in detail a number of *feasibility* issues fundamental to any message communication network supporting reliable concurrent computations. In particular, in section 2.2, we consider the issue of assuring communication deadlock freedom in spite of the introduction of arbitrary adaptive routes. The notion of a coherent data transfer protocol is introduced that allows for the adoption of voluntary misrouting. In section 2.3, we examine additional issues concerning network

progress, setting the stage for the development followed in sections 2.4 and 2.5, where we discuss methods to assure packet delivery and network access in detail. Finally, in section 2.6, we comment on a few remaining issues and summarize the chapter.

2.1 An Adaptive Cut-Through Model

In this section, we describe a simple model for the discussion of adaptive routing schemes in multicomputer networks. The model includes just enough structure to provide a context for the discussion of a number of fundamental issues in *distributed* systems. The following definitions develop the notation that will be used throughout the entire thesis.

Definition 2.1 A *multicomputer network*, M, is a connected undirected graph, $M = G(N, C)$. The vertices of the graph, N, represent the set of computing nodes. The edges of the graph, C, represent the set of bidirectional communication channels.

Definition 2.2 Let $n_i \in N$ be a node of M. The set, $C_i \subseteq C$, is the set of bidirectional channels connecting n_i to its neighbors in M. However, for the convenience of presentation, we shall occasionally distinguish between the input and output directions of a communication channel; in such case, it will be referred to specifically as either an *input* or an *output* channel.

Definition 2.3 The *width*, W, of a channel is the number of data wires across the channel. A *flit*, or *flow control unit*, is the W parallel bits of information transferred in a single cycle. The flit provides a natural unit for measuring the length of a packet.

Definition 2.4 Let $\mathcal{M} : N \times N \mapsto I$ denote a *metric* function from the set of ordered pairs of nodes to the set of nonnegative integers, such that $\mathcal{M}(n_i, n_j) = 0$ if and only if $i = j$, and such that $\mathcal{M}(n_i, n_k) \leq \mathcal{M}(n_i, n_j) + \mathcal{M}(n_j, n_k)$. An example of such a metric is the graph-theoretic distance function between any pair of nodes.

Definition 2.5 Given a metric function $\mathcal{M}$, a channel $c_j \in C_i$ that connects node n_i to node n_j is *profitable* with respect to node n_k if and only if $\mathcal{M}(n_i, n_k) > \mathcal{M}(n_j, n_k)$. In other words, forwarding a packet across a profitable channel will always reduce the packet's distance from its destination.

Definition 2.6 For each node $n_i \in N$, the *routing relation*, R_i, is the set of ordered pairs (n_k, c_j), where $n_k \neq n_i$ is a node and $c_j \in C_i$ is a profitable channel from n_i with respect to n_k. For convenience, we shall also use the notation R_{ik} to denote for each possible destination node, $n_k \in N$, its corresponding profitable channel set, from n_i.

Definition 2.7 The actual path a packet traverses while in transit in the communication network is referred to as the *trajectory* of the packet. Packet trajectories are identical to the packet routes in oblivious routing schemes but are *nondeterministic* in our adaptive formulation.

We assume the following:

- Long messages are broken into packets that are the logical data entities transferred across the network.
- Packets are of fixed length; *ie*, packet length = L, where L is a network-wide constant.
- Complete destination information is included in the header flit of length one in each packet.
- Message packets are forwarded in virtual ***cut-through*** style; *ie*, packets can immediately be forwarded once sufficient routing information has arrived to permit the corresponding routing decisions.
- Once a packet has started transmission across a channel, it cannot be interrupted until the transmission of the entire packet has completed.
- A message packet arriving at its destination is consumed. This is commonly known as the ***consumption assumption.***
- A node can generate messages destined to any other node in the network.
- Nodes can produce packets at any rate subject to the constraint of available buffer space in the network, and packets are source queued.
- Each node in the network has complete knowledge of its own routing relation.

Figure 2.1 presents our view of the structure of a node in a multicomputer network. Conceptually, a node can be partitioned into a computation subsystem, a communication subsystem, and a message interface. The computation subsystem consists of conventional processor and memory modules, whose internal structures do not concern us. For our purpose, the computation subsystem serves as the producer and consumer of the messages routed by the communication subsystem of the node. The message interface consists of dedicated hardware that handles the overhead in sending, receiving, and reassembling message packets. Internally, the communication subsystem consists of an adaptive control and a small number of message-packet buffers. Routing decisions are made by the adaptive control, based entirely on locally available information.

Complete knowledge of routing relations can be realized either by storing routing tables at each node, or by having a well-defined procedure to generate elements of the routing relations on the fly. Such procedures are particularly simple in highly regular network topologies such as the k-ary-n-cubes. The bidirectional channel assumption is different from the unidirectional channel assumption adopted in [11]. Having bidirectional channels allows the network to exploit locality in general message communication patterns. Another important property of the bidirectional channel assumption is that it assures an identical number of input and output communication channels in each node, irrespective of the underlying network topology. The ***fixed*** packet-length assumption is not essential, and can be replaced by a ***bounded*** packet-length assumption, *ie*, packet length $\leq L$, without invalidating any of our major results. It is adopted solely to simplify our subsequent exposition. Furthermore, engineering considerations also favor the fixed-length assumption over the bounded-length assumption, since the former is likely to admit much simpler hardware implementations.

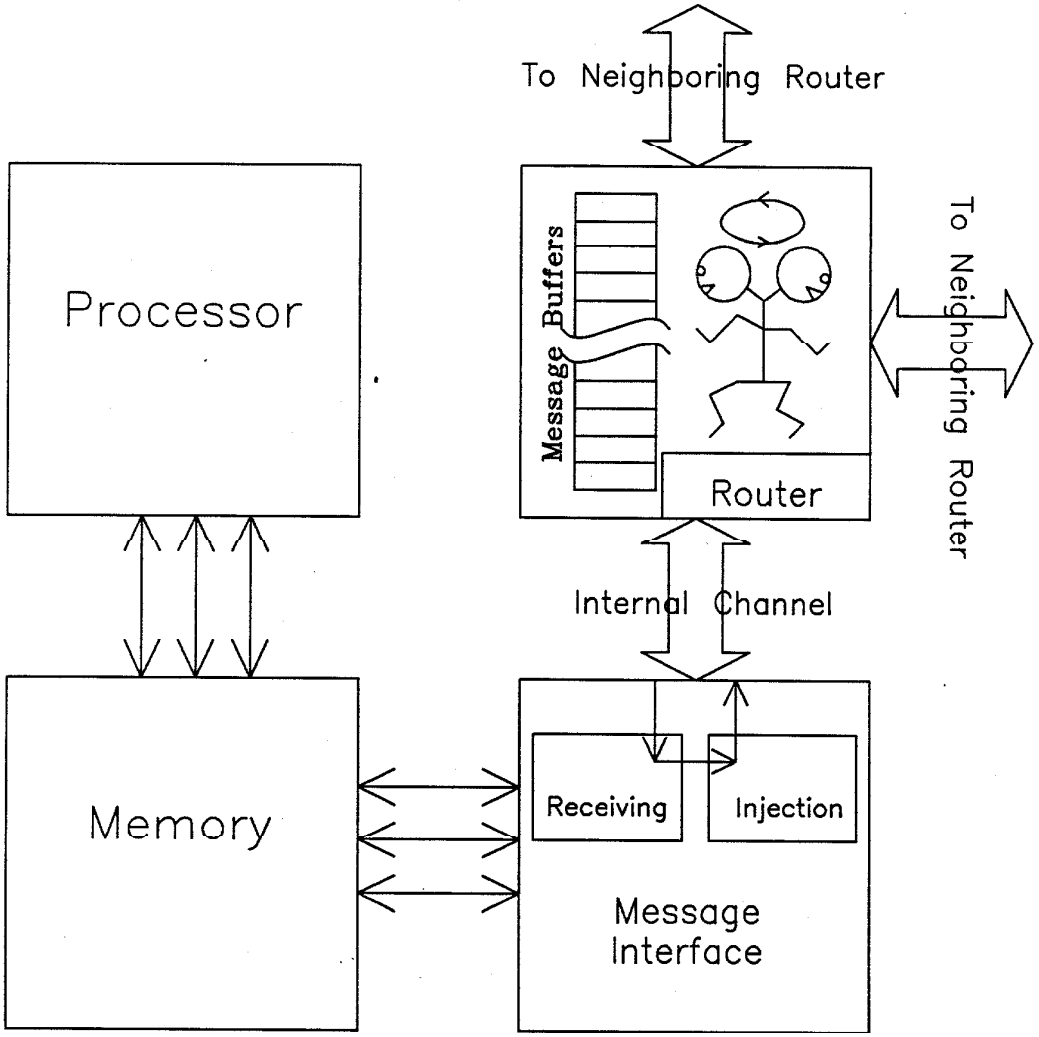

Figure 2.1: Structure of a Node

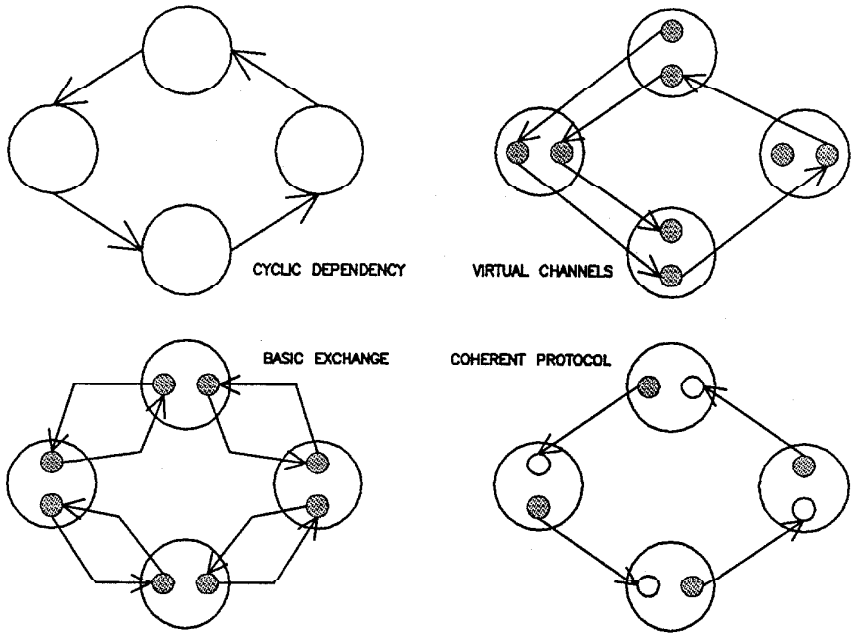

Figure 2.2: A Variety of Deadlock Prevention Techniques

2.2 Communication Deadlock Freedom

As it was mentioned at the beginning of the chapter, the ability to assure freedom from communication deadlock is essential to any practical routing schemes proposed for multicomputer networks. In this section, we shall examine the issue of communication deadlock in detail and demonstrate how voluntary misrouting can be applied to assure deadlock freedom.

Communication deadlock is caused generically by the existence of *cyclic* dependencies among communication resources along the message routes. Methods to prevent communication deadlock have been intensively researched [19,59], and many schemes exist; of these, the methods of structured-buffer pools [37] and virtual channels [12] are representative (see Figure 2.2). In essence, all of these methods approach the problem by remapping any dependency that is potentially cyclic into a corresponding acyclic dependency structure. These methods employ restructuring techniques that require information of a global, albeit static, character. The length from a source to a sink in these acyclic dependency structures is typically much longer than one. Depending on the size of the network and, hence, the length of these dependency chains, in general, one cannot put a fixed bound on the duration of the request-acknowledge transfer cycle across a channel, even though it is guaranteed to be finite.

In contrast, the simple technique of performing voluntary misrouting was suggested in [8] for synchronous networks that employ data exchange operations. Voluntary misrouting utilizes only local information, and is ***independent*** of network size and topology. It prevents deadlock by ***breaking*** the potentially cyclic communication dependencies into disjoint paths of unit length. In fact, a similar notion lies behind the ***packet exchange*** model described in the author's earlier report [41], where the cyclic dependencies are

broken and replaced by local cycles of length two. Voluntary misrouting can be applied to assure deadlock freedom in cut-through switching networks, provided the input and output data rates across the channels at each node are tightly *matched*. A simple way is to have all bidirectional channels of the same node operate *coherently* under the data-transfer protocol described in the next subsection.

2.2.1 The Coherent Channel Protocol

We now describe the coherent channel data-exchange protocol in detail. It is used to match the transfer rates across all channels of the same node. The protocol employs four control signals per channel, two from each of the communicating partners, and is completely symmetric between the partners. The signaling events for a channel $c \in C$ are defined as follows:

- R_o^c — output event to the communicating partner indicating that this node is Ready to accept another input flit from its partner. It also serves as an acknowledgment to its partner for the successful completion of the previous transfer cycle.
- R_i^c — input event from the communicating partner indicating that the partner is Ready to accept another output flit from this node. It is also an acknowledgment from the partner for the successful completion of the previous transfer cycle.
- V_o^c — output event to the communicating partner indicating that the data flit values currently held at the output channel of this node are Valid and its partner should latch in the held values.
- V_i^c — input event from the communicating partner indicating that the data flit values currently asserted at the input channel of this node are Valid and the node should latch in the held values.

Since the coherent channel protocol is used to match the transfer data rates across all channels of a node, it is natural to generate the same handshaking output control signals R_o^c, and V_o^c for all the output channels of the same node. In other words, for node n_k, we have:

$$\begin{aligned} R_o^c &\equiv R_o, \qquad \forall c \in C_k \\ V_o^c &\equiv V_o, \qquad \forall c \in C_k \end{aligned}$$

We now proceed to define our handshaking protocol across channels of a node, $n_k \in N$, in a CSP-like notation [33]:

$$*[\ \ R_o,\ \ [\forall c \in C_k,\ R_i^c];\ \ \text{apply out data};\ \ V_o,\ \ [\forall c \in C_k,\ V_i^c];\ \ \text{latch in data}\ \ ]$$

where R_o and V_o denote, respectively, the unique outgoing Ready and data Valid signaling event to all neighbors. Briefly, the protocol works as follows: The pair of R_o, R_i^c events is used to synchronize the communicating nodes across the channel into the common state of being ready to accept another round of data exchange with its partner. Similarly, the V_o, V_i^c pair of events serves to synchronize the nodes into the common state of being able to latch in a valid input data flit from its partner. The handshaking

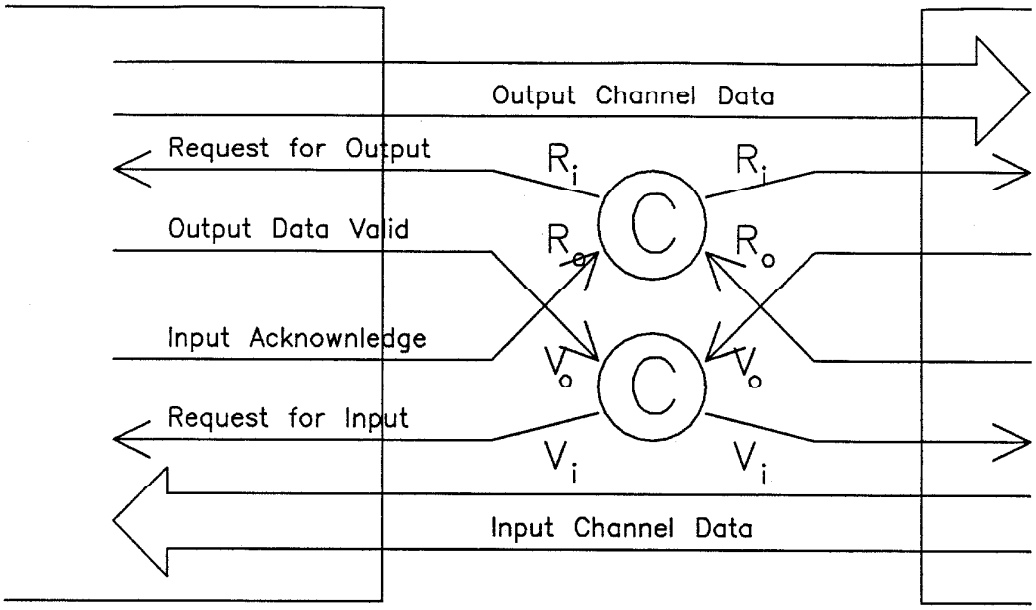

Figure 2.3: A Two-Phase Signaling Realization of the Coherent Protocol

events R_o, R_i^c interlock with the events V_o, V_i^c to guarantee the stability and strict alternation of each other. This strict alternation of input and output events suffices to match the transfer rates of opposite directions across a channel. On the other hand, matching of output data rates across all channels of the same node is immediate, since they all share the same output control signals, R_o and V_o. In contrast, matching of input data rates across all channels of the same node is enforced through the *synchronized wait* for the R_i^c and V_i^c signaling events from every neighbor of the node. These notions will be formalized explicitly in the next subsection. The initial state of a channel after network startup has both directions of the channel ready to accept a new data flit. Notice that the operation of the exchange protocol with this defined initial state proceeds in a *demand-driven* fashion that is different from the usual *supply-driven* request-acknowledge cycle. Figure 2.3 shows a possible conceptual realization of the exchange protocol under the two-phase signaling convention popular for off-chip communication [48].

Observe that under cut-through switching, a packet can span many different channels. An outgoing channel occupied by a packet may not be able to assert V_o until after valid data has been asserted by the corresponding incoming channel occupied by the packet; this induces matching of data rates across the two occupied channels. The notion of coherency introduced here is a natural way to accommodate such potential dependencies among the various channels of a node under cut-through switching. Another notion that arises naturally is that of a *null flit*. The coherent protocol defined above enforces an equal transfer rate along opposite directions across each physical channel at all times. To effect a transfer of data in one direction of a channel while the opposite direction is idle, the receiving partner is required to transmit a null flit in order to satisfy the convention dictated by the exchange protocol.

2.2.2 Properties of the Coherent Protocol

The coherent protocol exhibits a number of interesting and useful logical properties that we shall now investigate. These properties will be exploited later to establish our fundamental results on packet deliveries and injection fairness.

Let $N(R_i^c)$, $N(R_o)$, $N(V_i^c)$, and $N(V_o)$ denote, respectively, the total number of occurred R_i^c, R_o, V_i^c, and V_o events across a channel $c \in C_k$ since initialization. The following invariant conditions are satisfied by a bidirectional channel that follows the coherent protocol under the defined initial conditions:

$$\begin{array}{lrcccl} (I1) & 0 & \leq & N(R_o) - N(V_i^c) & \leq & 1 \\ (I2) & 0 & \leq & N(R_i^c) - N(V_o) & \leq & 1 \\ (I3) & 0 & \leq & N(R_i^c) - N(V_i^c) & \leq & 1 \\ (I4) & 0 & \leq & N(R_o) - N(V_o) & \leq & 1 \end{array}$$

The first two inequalities simply express *causality* conditions implied by our demand-driven protocol: A node transfers valid data to an output channel only after the receiving partner has indicated its readiness to accept. The last two inequalities express conditions that are direct results of the strict alternation of ready and data-valid events and the specified initial state. In essence, it requires the communicating nodes to send their data out, *ie*, increment $N(V_o)$ and $N(V_i^c)$, respectively, only after the nodes themselves have indicated their willingness to accept new data from their respective partners. Together, these conditions result in a cycle-by-cycle exchange of data flits between the communicating nodes connected by the channel. Subtracting the two pairs of invariant conditions from one another, we obtain the following pair of invariant conditions, $\forall c \in C_k$, of a node n_k:

$$\begin{array}{lrcccl} (I5) & 0 & \leq & |N(R_i^c) - N(R_o)| & \leq & 1 \\ (I6) & 0 & \leq & |N(V_i^c) - N(V_o)| & \leq & 1 \end{array}$$

The tight matching of the data transfer rates of the opposite directions of a channel has been made explicit by these two invariant conditions. The strategy is to equalize the number of data-flit transfer operations back and forth along the channel to within a difference of at most one at all times. This is equivalent to restricting the communicating partners to within a *synchronic distance* of one from each other in the *Petri Net* theoretic sense [45]. Similarly, according to our initial set of invariants, we have for different channels of the same node n_k:

$$\begin{array}{llrcccll} (I2) & & 0 & \leq & N(R_i^c) - N(V_o) & \leq & 1 & \forall c \in C_k \\ \Rightarrow & (I7) & 0 & \leq & |N(R_i^{c_1}) - N(R_i^{c_2})| & \leq & 1 & \forall c_1, c_2 \in C_k \\ (I1) & & 0 & \leq & N(R_o) - N(V_i^c) & \leq & 1 & \forall c \in C_k \\ \Rightarrow & (I8) & 0 & \leq & |N(V_i^{c_1}) - N(V_i^{c_2})| & \leq & 1 & \forall c_1, c_2 \in C_k \end{array}$$

In other words, we are assured that the data rates across different channels of the same node are matched to each other at all times.

Observe that according to the protocol, each node inside the network carries out the required handshaking with its nearest neighbors locally without any global knowledge of the states of nodes farther away. In fact, nodes that are of distance d apart in the network are of synchronic distance $\lceil d/2 \rceil$ from each other; *ie*, they may have their respective total

number of cycle completions differ by as much as $\lceil d/2 \rceil$. To see this, let node n_0 be the source, and consider a path of length d joining nodes n_k, $k = 1, 2, \ldots, d-1, d$, which are, respectively, distance i away from n_0. According to the invariant condition $(I7)$, we have $0 \leq |N(R_i^{c_1}) - N(R_i^{c_2})| \leq 1$, $\forall c_1, c_2 \in C_k$ of the same node n_k. Therefore, nodes n_{k-1} and n_{k+1}, which are both neighbors of n_k, can have their number of completed cycles differ by at most 1. Applying this fact to nodes $n_0, n_2, n_4, \ldots$ makes it clear that the cumulative difference between n_0 and n_d can be as much as and no more than $\lceil d/2 \rceil$. Global synchrony does not exist and there is no global synchronization signal to be distributed over the entire network. As a result, a network following the protocol is potentially infinite in size, or, realistically, ***arbitrarily extensible.***

While there is no global synchrony of events nor any ***unique global clock*** over the entire network, the local synchrony enforced by the coherent protocol allows one to define a set of ***local clocks*** that are compatible with each other. More precisely, we define the value of a discrete local clock T_k of a network node $n_k \in N$ as the total number of R_o signaling events occurred after initialization of the node. Intuitively, each R_o event signals both the completion of the previous transfer cycle and the beginning of the next cycle, when new decisions have to be made within each node to determine if data are to be forwarded to neighboring nodes or to be buffered internally. Hence, one is justified in regarding the R_o events as defining local clock cycles with each clock cycle corresponding to a routing epoch.

Let each message packet, p_m, traveling inside the network, keep count, A_m, of the total number of R_o events that p_m has observed since its injection. Two local clocks, T_j and T_k, of nodes n_j, $n_k \in N$, respectively, are ***compatible*** if a packet, p_m, traveling in the network starting from node n_j at local time $T_j = T_j^s$ with count $A_m = A_m^s$ and ending at node n_k at local time $T_k = T_k^e$ with count $A_m = A_m^e$ will find, upon its arrival at n_k, an agreement:

$$T_k^e - T_j^s = A_m^e - A_m^s$$

To see that the defined set of local clocks is indeed compatible, observe that we only have to establish its validity when our packet, p_m, is being forwarded to the next node along its trajectory during a protocol cycle; otherwise, the count recorded by the packet simply tracks the local time of its resident node. During the actual transfer of p_m from one node to another, the sender signals the beginning of the transfer with a V_o event, thus incrementing $N(V_o)$. The coherency constraints dictate that $N(R_o) = N(R_i^c)$, $\forall c \in C$ at the sender side at this point; hence, the sender and all its neighbors must have identical clock values. When the receiver signals the completion of transfer with its own R_o event, p_m is already at the receiver side. Therefore, both the receiver's local clock value and the count recorded by p_m are incremented, preserving the agreement between packet count and the local clock value. Compatibility of local clocks follows by induction.

As a result of the compatibility of the defined set of local clocks, the recorded count values are not only consistent with the local clocks, but also consistent with each other. Specifically, let $\Delta A = A_m - A_n$ be the difference in the recorded count values of packets p_m and p_n upon a rendezvous: The value ΔA will remain identical upon all future rendezvous of the same packets p_m and p_n. In other words, we are justified in defining the value A_m as being the *age* of the packet p_m. The existence of a well-defined age for packets in a coherent network allows us to define meaningful dynamic priorities that

are essential in establishing packet-delivery guarantees. Furthermore, the basic protocol cycle provides a natural partition of continuous physical time into a sequence of discrete epochs upon which local routing decisions are made.

2.2.3 Deadlock Freedom

In the previous subsections, we have described the coherent data-transfer protocol for the physical channel, and examined its various properties in detail. In this subsection, we establish deadlock freedom assurance for coherent networks that apply voluntary misrouting as the means to prevent buffer overflow. To proceed, observe that routing under the cut-through switching model imposes the following integrity constraints:

1. Packets must always be forwarded to neighbors with their header flits transmitted first. In particular, voluntary misrouting of any internally buffered packet must start from the header flit of the selected packet.

2. Once the flit stream of a packet has been assigned a particular outgoing channel, the assignment must be maintained for the remaining cycles until the entire packet has been transmitted.

These constraints exist because all of the necessary routing information of a packet is encapsulated in the packet header. Interrupting a packet flit stream mid-transfer would render the latter part of the packet undeliverable. For a communication network that follows the coherent protocol, assurance of freedom from communication deadlock globally over the network is equivalent to assurance of *progressive* protocol execution locally at each node. The notion of progressive protocol execution can be quantified in terms of the total number of completed protocol cycles. In other words, assurance of communication deadlock freedom is translated into assurance of eventual increase in the total number of completed cycles over *every* node of the network. In particular, it is sufficient to show that each node can ***independently*** complete each transfer cycle and initiate a new one, without violating the stated constraints. To see this, take any snapshot of the network and observe that nodes having the ***minimum*** number of completed cycles will eventually increase their numbers, and, hence, the global minimum, provided the above constraints can always be satisfied. This in turn implies that the number of completed cycles in every node must also eventually increase. We now show that as long as we have an equal number of input and output channels per node, a condition that is satisfied readily by our bidirectional channel assumption, we can always satisfy the stated logical requirements, and, hence, assure freedom from communication deadlock.

Theorem 2.1 Let M denote a coherent multicomputer network where each node has an equal number of input and output channels. If M employs voluntary misrouting to prevent potential buffer overflow, then it is free from deadlock.

Proof. We need to show that buffer overflow can always be prevented by misrouting without violating the cut-through switching integrity constraints. We proceed with a counting argument: Let d denote the number of channels at a node. During a protocol cycle, there may be as many as $n^* \leq d$ new data flits arriving at the input channels. A fraction of these, $0 \leq n' \leq n^*$, are new header flits; the remaining n^*-n' are payload flits of arriving packets. Of these payload flits, a fraction of them, $0 \leq n'' \leq n^*-n'$, belong to

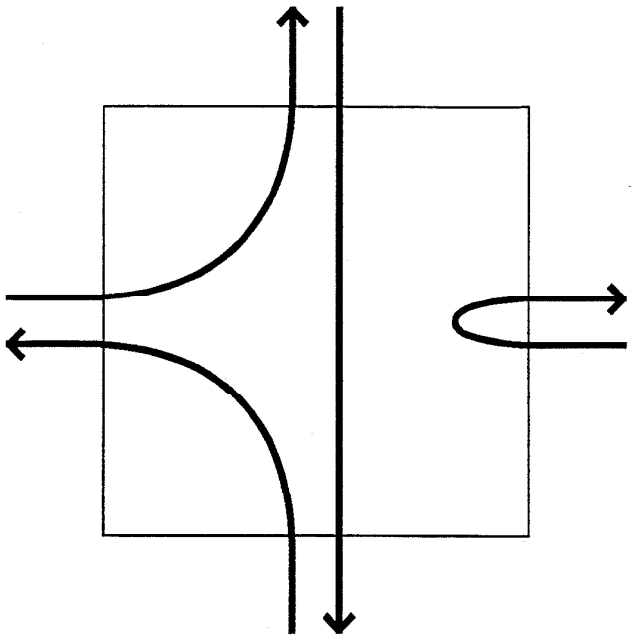

Figure 2.4: Deadlock-Free Routing under the No-Blocking Convention

packets that have already been assigned output channels, and the remaining $n^*-n'-n''$ flits belong to waiting packets that are buffered inside the node. Therefore, the node has at least a total of $n'+(n^*-n'-n'')$ header flits that are eligible for immediate routing. Hence, in the following cycle, a node can find at least $n'+(n^*-n'-n'')+n'' = n^*$ flits that can be transmitted or misrouted without violating the cut-through switching integrity constraints. This assures that no buffer overflow will occur. The node can always complete its protocol cycle and initiate the next protocol cycle; hence, the network is free from deadlock. ■

Since the validity of the above proof does not depend on a node's storage capacity, deadlock freedom is established independent of the amount of available buffer space. The simple criterion of having an equal number of input and output channels is sufficient to assure deadlock freedom for a coherent network. Figure 2.4 depicts a possible scenario when there is no internal storage for a node with four channels. Any packet that arrives is immediately shuttled away through one of the output channels. In practice, additional buffers are needed in order to inject packets into the network, and to improve the network performance. The reader may have noticed from the above proof that although deadlock freedom is guaranteed as long as the number of input channels is equal to the number of output channels, there is some real danger of an inability to deliver packets. This fact will be discussed in much greater detail in the following section.

2.3 Potential Lack of Progress

The coherent exchange model eliminates the possibility of deadlock in the adaptive cut-through switching formulation by guaranteeing the transfer of data flits at the node-to-

node level. Packet transfer at the source-to-destination level, however, is not immediate. In particular, a network can run into a *livelock*. Consider the sequence of routing scenarios depicted in Figure 2.5 for a bidirectional ring consisting of eight nodes and eight packets. Eack of the packets consists of four data flits that span multiple channels and internal buffers. Suppose the nodes employ the following simple, deterministic, packet-to-channel assignment rule: Whenever two incoming packets both request the same outgoing channel, the packet from the clockwise neighbor always wins. Given that, initially, nodes A, C, E, and G each receive two packets destined to nodes that are, respectively, distance two from them in the clockwise direction, then after four routing cycles, the packets are all back to where they started! In other words, the network enters a livelock cycle. This example illustrates how packets can be forever denied delivery to their destinations even in the absence of communication deadlock.

Channel-access competitions are, however, not the only type of conflict that can lead to livelock. Consider the situations depicted in Figure 2.6 for the same bidirectional ring network. The traffic patterns are coincidental in such a way that none of the packets will ever have a chance to select its own output channel; rather, at every node, each packet must be forwarded along the only remaining channel, in compliance with the voluntary misrouting discipline, in order to avoid deadlock. It is clear that no matter what assignment strategy one chooses, it is impossible to break this kind of livelock without adding extra buffers per node. In other words, additional measures and resources have to be introduced in order to assure progress, *ie*, delivery of packets, in the network.

We distinguish between several different types of progress guarantees that should be present in a multicomputer communication network. First, every individual packet in transit across the communication network should be guaranteed delivery within a bounded finite period. A weaker form guarantees that *some* packets in transit across the network will be delivered within a bounded finite period. In this weaker form, we are assured of some global progress inside the network, but can make no corresponding statement about each and every packet.

The second type of progress guarantee concerns the initial injection of packets into the communication network. Consider the pathological situation depicted in Figure 2.7, where a node situated at the cross point of continuous heavy traffic can be prevented from sending packets into the network for an unbounded amount of time. This happens because a node has only bounded finite capacity and the coherent protocol dictates a strict balance of incoming and outgoing traffic across the channels of a node. Ideally, every node that has a packet queued for entry into the network should be guaranteed access into the network within a bounded finite period.

There are a number of other related considerations besides these two types of progress guarantees. Depending on the intended programming model and semantics to be supported by the multicomputer network, it may be necessary to guarantee preservation of message order between any pair of communicating nodes [47]. Usually, this requirement is satisfied at a higher level of abstraction than we have discussed here. For example, one simple way to enforce message order preservation is through the use of *acknowledgments*. A sender will not send out another message until the arrival of its previously sent message has been acknowledged. Another standard way is to store within every receiving node, a message arrival count for each potential sending node.

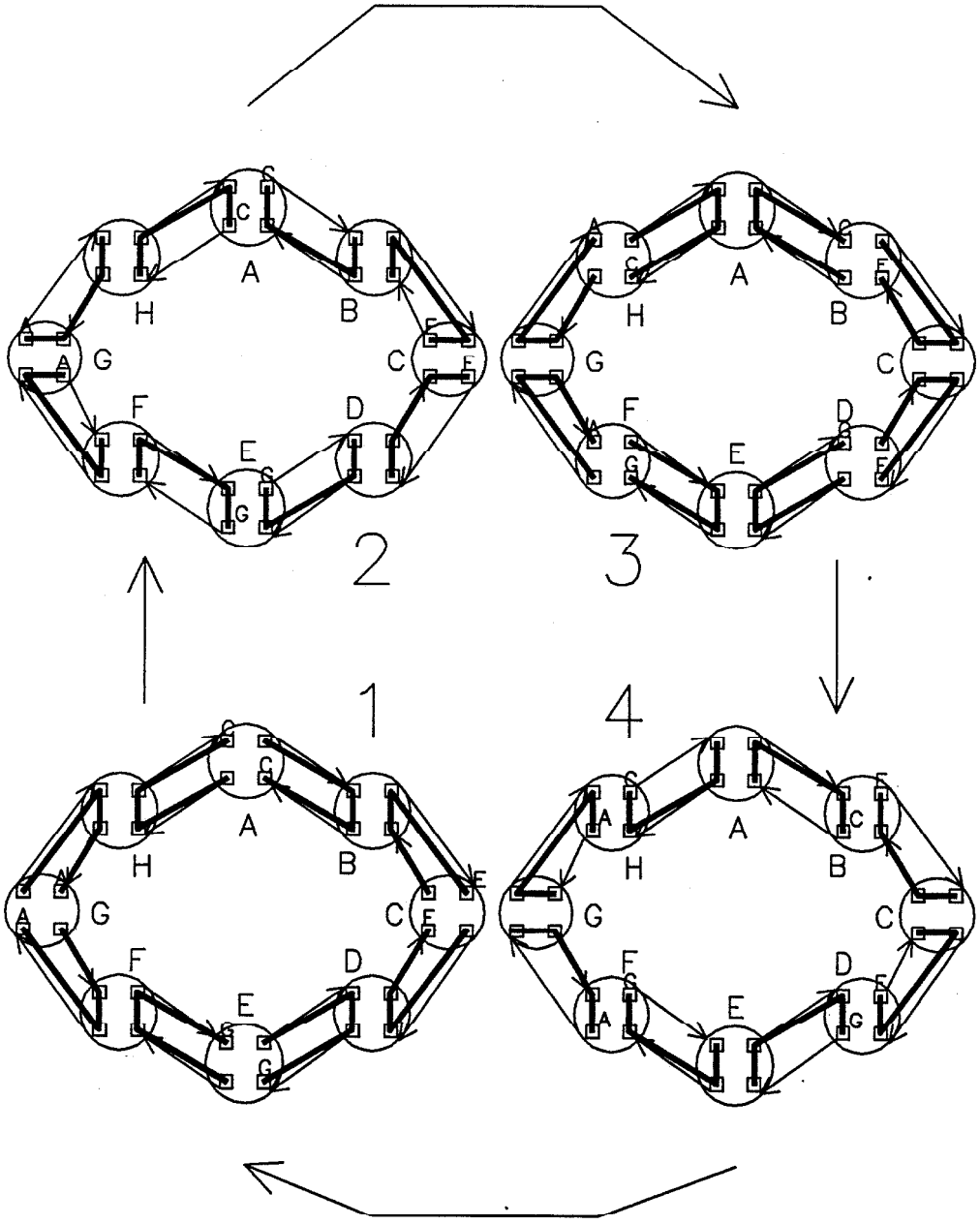

Figure 2.5: Livelock due to Bad Routing Assignments

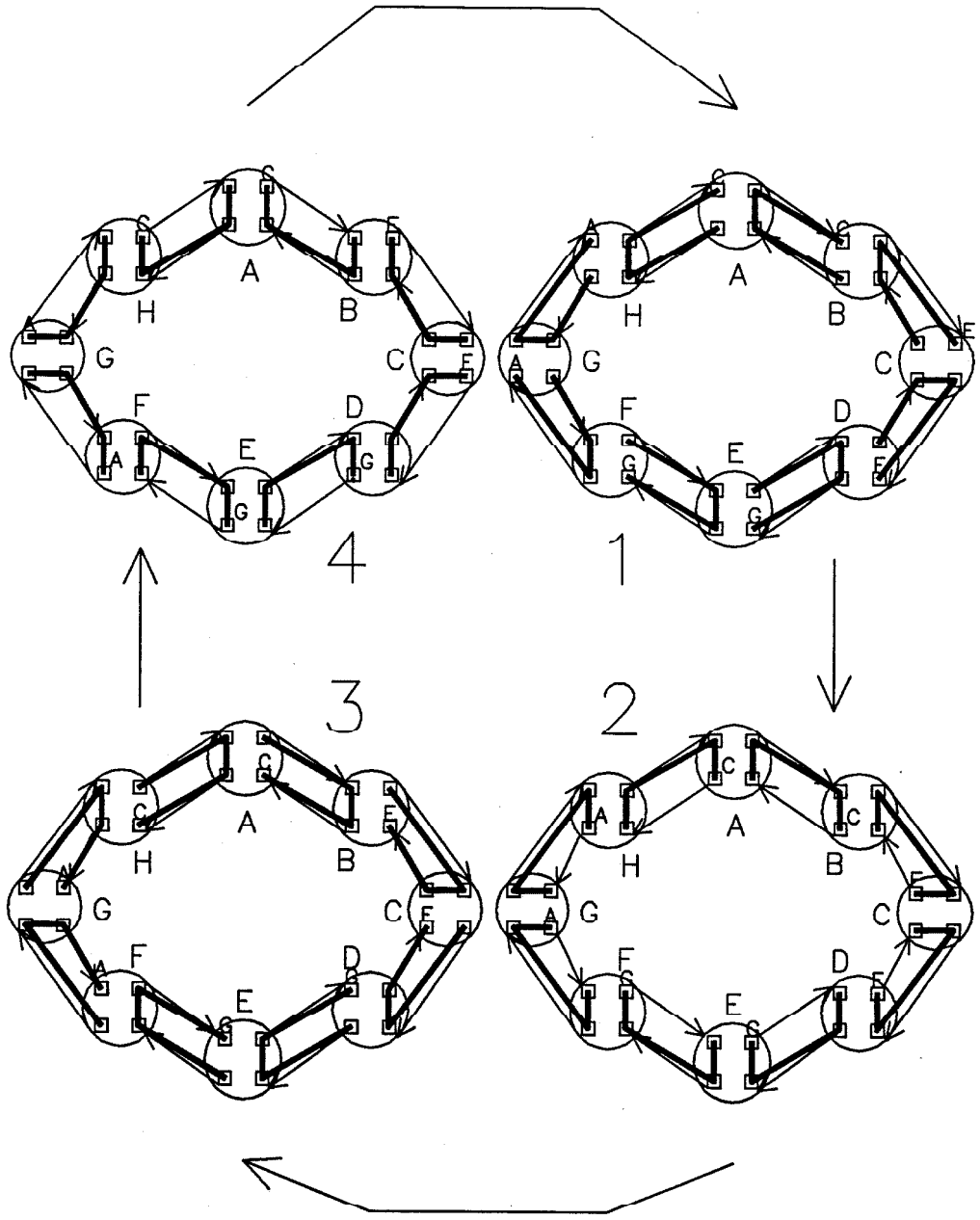

Figure 2.6: Livelock due to Lack of Routing Assignments

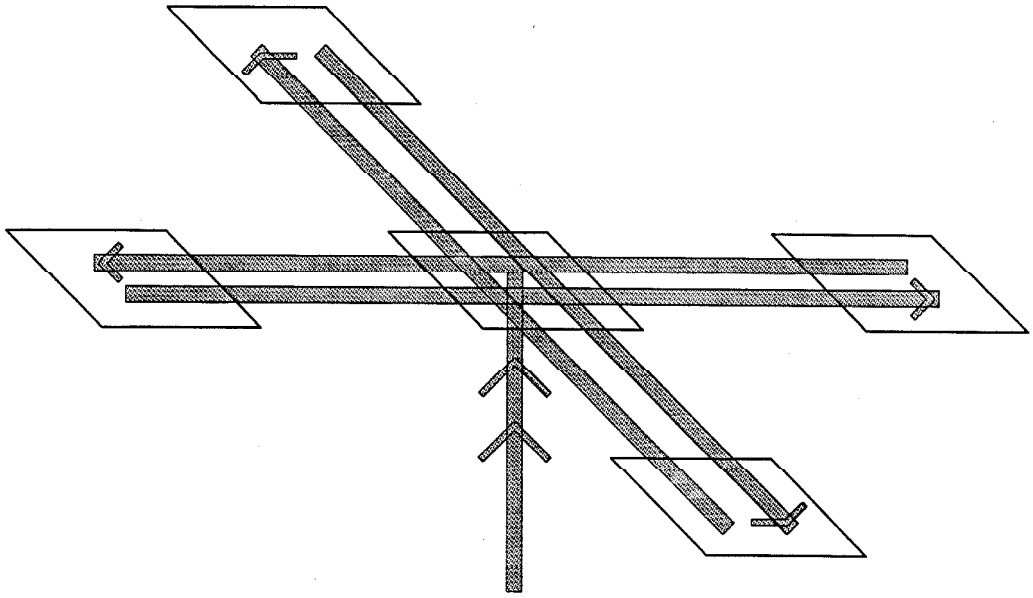

Figure 2.7: Inability to Inject Packets

The arrival counts allow the nodes to determine the *sequence* number of the next arrival message in correct order. Unfortunately, these schemes suffer such obvious drawbacks as longer message latencies due to the wait for acknowledgments, or extravagant use of memory due to the storage of arrival counts. In this respect, *actor* languages [1] that do not assume message order preservation in their basic language semantics, appear to be ideal candidates for being supported in networks implementing the adaptive formulation.

Another closely related issue concerns the transmission of multipacket messages. In an adaptive routing scheme, the different packets of a multipacket message may be forwarded along different routes and will not necessarily arrive at the destination node at the same time. If a message can only be consumed at the destination node after the whole message has been received, then *reassembly deadlock* can arise, where packets of different messages fill up all the available buffering space of a node and prevent any of the partially delivered messages from being reassembled [56]. Here, we shall assume that message reassembly is accomplished in the node memory by the message interface hardware. The node memory is normally much larger than the relatively small number of buffers available inside the router; hence, reassembly deadlock is very unlikely. In any case, its effective resolution requires coordination across higher levels of abstraction. In the following sections, we shall focus on describing basic schemes for assuring individual packet injection and delivery in networks that employ the adaptive cut-through switching formulation.

2.4 Packet-Delivery Guarantees

The preceding discussion establishes the need to assure packet injection and delivery. In this section, we address the problem of devising general schemes for assuring individual packet delivery. General schemes for assuring individual packet injection will be discussed in the next section.

In general, given a fixed network topology and the set of routing relations in each node of the network, there could conceivably be a number of different ways to assure eventual packet delivery. For example, one may approach the problem by picking a packet-to-channel assignment scheme that exploits the underlying network topology in assuring packet delivery. For instance, in our previous ring network, by agreeing only to send packets out in the clockwise direction on a first-in-first-out basis, which is a restricted form of packet-to-channel assignment, we can actually assure individual packet delivery. It is interesting to note that this technique can be generalized: Observe that our networks, which have equal numbers of input and output channels per node, guarantee the existence of an *Eulerian* circuit. By routing packets along this circuit, one can indeed guarantee delivery of every individual packet. However, this solution is clearly unacceptable because it is oblivious, and does not take advantage of the rich connectivity inherent in these networks. In any case, *ad hoc* schemes are extremely difficult to devise and analyze when one considers more complicated topologies.

The example in Figure 2.5 of the previous section demonstrated that livelock cycles in cut-through switching can result from bad choices of routing assignments. On the other hand, the example in Figure 2.6 shows that certain other livelock cycles exist that are impossible to break regardless of the assignment strategy utilized. Packets are trapped in these cycles precisely because they never have the chance to apply and then follow the chosen assignment strategy. To assure eventual packet delivery, we shall isolate these two possibilities and address each separately. In particular, we shall provide a sufficient number of buffers at each node so that the adaptive control will always be able to guarantee that at least one packet can stay behind and avoid being misrouted, if so determined by the assignment strategy. By having a sufficient number of buffers, competition for access to profitable channels is then transformed into a competition for the right to stay behind and wait until the winner's profitable channel becomes available, at which time it will be forwarded. This way, winners that have been chosen by the assignment algorithm will have the chance to follow the actual paths determined by the routing relations. In other words, assurance of packet delivery will then be reduced to that of picking *consistent* winners across the network.

In subsection 2.4.1, we describe a simple but realistic buffer structure, and derive the minimum required number of buffers under the assumed buffer-allocation discipline, thus demonstrating the existence of such a solution using a bounded number of buffers. In order to guide the choice of consistent winners, in subsections 2.4.2 and 2.4.3 we present the simple idea of a packet *priority* scheme as a general method for generating assignments that assure individual packet delivery.

2.4.1 Buffering Discipline and Requirement

In order to discuss the buffer storage requirements, it is necessary to state the assumptions regarding the buffer structures and allocation schemes that are employed. In

general, the assumptions will depend on the implementation medium, and will directly affect the derived results. The assumptions listed below represent a reasonable compromise between organization simplicity and utilization efficiency. The main objective is to demonstrate the existence of a solution, that will avoid livelock cycles, and that uses only a bounded number of internal buffers. We assume the following:

1. Storage is divided into buffers of equal size; each is capable of holding an entire message packet.

2. Each buffer has exactly one input and one output port; this permits simultaneous reading and writing. A good example is a *FIFO* queue of length L.

3. Except as stated below, a buffer can be occupied by only one packet at a time. Oftentimes a packet may not fill its entire buffer, as in the case of a partial cut-through. Such a packet occupies both the input and output ports to the buffer.

4. A buffer can be used temporarily to store two packets at a time if, and only if, one of them is leaving through the output port connected to an output channel, and the other is entering through the input port connected to an input channel.

Recall that under the cut-through switching model, when a packet arrives at an intermediate node, it is possible that all of its profitable outgoing channels are busy. With the presence of empty internal buffers, the incoming packet can be temporarily stored and then forwarded as soon as the *first* profitable outgoing channel is available. However, under heavy continuous incoming packet traffic, a node may be forced to misroute some of its buffered packets in order to prevent buffer overflow. Our objective is to show that by providing sufficient buffers, it is possible to allow any buffered packet which is chosen by the assignment algorithm, to stay and avoid being misrouted. To derive the required number, let b and d denote, respectively, the number of buffers and channels, *ie*, the *degree*, at each node. First, we observe that given the above buffering discipline, we must have $b \geq d$. To see this, assume that $L \gg d$, and consider the following sequence of events at a node with all buffers initially empty: At cycle $t = 0$, a packet, P_0, arrives and is forwarded to its requested output channel, c^* at cycle $t = 1$. Then, at cycles $t = L-d$ up to $t = L-2$, a total of $d-1$ packets, P_i, $i = 1, \ldots, d-1$, arrive one after another in these $d-1$ consecutive cycles, all requesting the same output channel c^*. Finally, at cycle $t = L+2$, another packet, P_d, arrives, requesting the same channel c^*. The worst case happens when the assignment algorithm always favors the latest-arriving packet by requiring it to stay and avoid misrouting, while having each packet occupy a distinct buffer. Given the above arrival sequence, at cycle $t = L+1$, packet P_{d-1} will be forwarded through c^*, which now becomes idle. As a result, each packet from P_1 up to P_d would have to be temporarily stored as it arrives. Since each packet must be allocated to a distinct buffer, we must have $b \geq d$. We now show that having $b = d$ buffers is also sufficient.

Theorem 2.2 Let M be a coherent network where each node has b packet buffers inside the router operating under the stated assumptions. Then $b = d$ buffers per router is necessary and sufficient to always allow at least one packet, chosen arbitrarily by the assignment algorithm at each node, to escape misrouting.

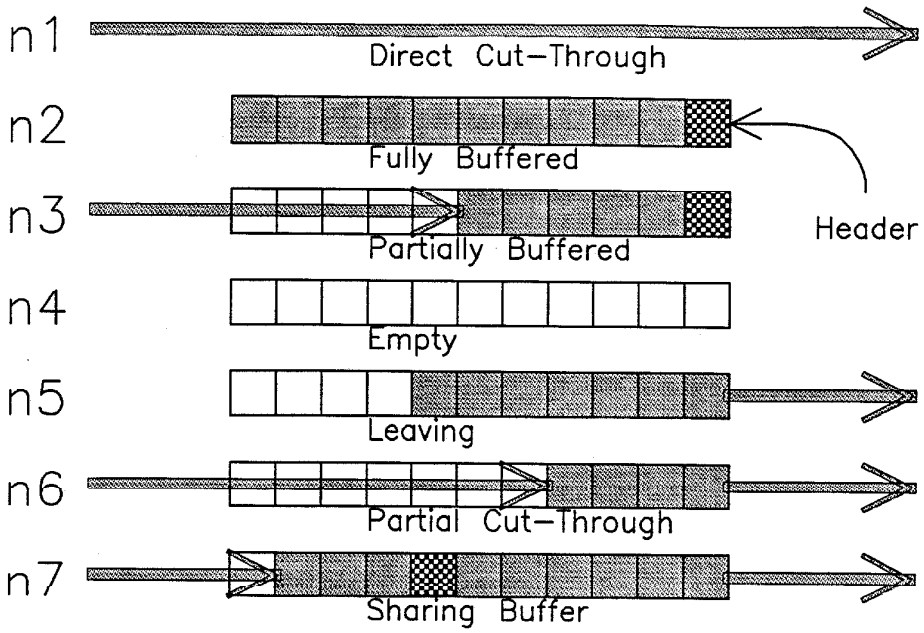

Figure 2.8: Accounting of All Possible Cases of Buffer Allocation

Proof. Necessity follows immediately from the preceding discussion. We proceed to establish sufficiency through a counting argument. Observe that a node is required to consider misrouting of packets in the next cycle only when there are new packets arriving at the current cycle. Figure 2.8 depicts an accounting of all possible cases of buffer allocation at the beginning of any such routing cycle. Let n_1 up to n_7 denote, respectively, the number of packets or buffers in each case; and n_0 denote the number of newly arrived packets. Then, for inputs, we have $n_0+n_1+n_3+n_6+n_7 \leq d$; for outputs, we have $n_1+n_5+n_6+n_7 \leq d$. To simplify the counting argument, let us assume for the moment that $n_0 = 1$. Let P^* denote the privileged packet chosen by the assignment algorithm to stay behind and avoid misrouting in the following cycle. P^* must be either the newly arrived packet or an already buffered packet. If P^* is a buffered packet, then the newly arrived packet either finds an idle output channel to directly cut through the node; or else we must have $n_1+n_5+n_6+n_7 = d \Rightarrow n_5 \geq n_0+n_3$, which, in turn, implies that there will always be an available buffer ready to accept it. On the other hand, if P^* is the newly arriving packet, then either $n_4+n_5 > 0$, and, hence, there is a buffer ready to accept it; or else we must have $n_2+n_3+n_6+n_7 = b = d$. This, together with the above inequality on inputs, $\Rightarrow n_2 \geq n_0+n_1 \Rightarrow n_2 > 0$. Furthermore, $n_0 > 0 \Rightarrow n_1+n_6+n_7 < d$. In other words, the packet will be able to find at least one buffer with a full idle packet as well as an idle output channel to misroute this idle packet and thus make room for itself. This establishes the validity for single-packet arrivals. Finally, repeated applications of the above argument then establish the validity for multiple-packet arrivals, and, hence, the sufficiency condition. ■

As demonstrated clearly from the above proof, the trick in allowing the escape from misrouting for any arbitrarily chosen packet is to provide at least a critical, minimum num-

ber of buffers that is sufficient to assure either that empty buffers still exist, or that all buffers have been occupied, and, hence, there is some other packet that can be misrouted instead. The particular minimum number required depends on the adopted buffering structure and discipline, and adding more buffers per node will allow the assignment algorithm to operate with more flexibility and perform better. Having established the buffering requirements, in the following subsections we proceed to demonstrate how to resolve channel-access conflicts and pick consistent winners.

2.4.2 Static Environment

We start with a *static* finite system where no new packet may be injected into the network until all present packets have been delivered to their respective destinations. In this case, there are only a finite number of packets inside the network. These packets compete among themselves if necessary to gain access to node buffers and communication channels that will bring them closer to their respective destinations. To each packet, we now assign a fixed priority chosen from a linear order such that no two packets have the same priority. For example, if no two packets inside the network are generated from the same source node, then the *source node address* of each packet can be used as the priority of that packet, assuming that the node addresses form a linear order. In other words, we have:

$$P_1 > P_2 \Longleftrightarrow N_1 < N_2 \tag{2.1}$$

where P is the priority of a packet and N its source node address. Resolution of channel-access conflicts are now based on the priorities of the competing packets, so that the ***highest*** priority packet among the competitors is always the winner. We now show that this is sufficient to guarantee eventual delivery of every single packet inside the network.

Lemma 2.3 A packet-to-channel assignment strategy that observes the packet-priority ordering defined in equation 2.1 guarantees eventual delivery of all the packets inside a static finite system.

Proof. During any routing cycle, the packet with the highest priority will always win in its competition; hence, it will eventually reach its destination and be removed from the network, thus reducing the total number of packets. Upon delivery, the packet with the next highest priority then takes its place and the above argument is applied again. Since we have only a finite number of packets inside the network, we are guaranteed that eventually all the packets will be delivered. ∎

The fixed-priority ordering defined in equation 2.1 assumes that each node has injected at most one packet into the network in our static environment. This is unnecessarily restrictive, and we present a different priority-assignment scheme that removes this restriction. This alternate scheme is also interesting in its own right in that it allows us to look at the routing actions from a different point of view, one that leads to the weaker form of a global network progress guarantee for a dynamic environment.

The process of forwarding a packet toward its destination can be viewed as a sequence of actions performed to reduce the packet's ***distance from destination***, provided that the set $\mathcal{R} = \{R_i\}$ of routing relations is defined in terms of an underlying metric of the network. In which case, as the result of a channel-access conflict, the winner will be

routed along a profitable channel, hence decreasing its distance from the destination. The losers, depending on whether they are routed away along the remaining unprofitable channels, may or may not increase their distance from destination. Ideally, one would prefer a strict monotonic decrease of distance to destination for each packet inside the network. As this is impossible under our adaptive model, the alternative is to ensure monotonic decrease over a sequence of exchanges involving a multiple number of packets. This can be achieved by giving higher priority to packets with shorter distances from destination over those with longer distances. In other words, we are motivated to define the distance-priority ordering as follows:

$$P_1 > P_2 \Longleftrightarrow D_1 < D_2 \tag{2.2}$$

where D is a packet's distance from destination. We now show that this alternate form also guarantees eventual individual packet delivery in a static environment.

Lemma 2.4 A packet-to-channel assignment strategy that observes the priority ordering defined in equation 2.2, together with the set, $\mathcal{R}$, of *metric-based* routing relations, guarantees eventual delivery of each individual packet inside a static finite network.

Proof. At the beginning of a routing cycle, let $D > 0$ be the minimum packet distance from destination. During this cycle, a packet with distance D competes with other packets for channels leading to its destination. If it wins the competition, it will be forwarded along a profitable channel within L cycles. It it loses, it must be to another packet also distance D away from its destination, according to the defined priority. In both cases, the minimum distance is reduced to $< D$ within L cycles. Therefore, D will eventually be reduced to zero, in which case a successful packet delivery occurs and the above argument can be applied again to assure repeated deliveries. This establishes livelock freedom. ∎

In some networks, such as the k-ary-n-cubes or meshes, the address of the destination node of a packet is completely specified by the relative distances the packet has to travel in each dimension. Hence, the relative distance from destination in such networks can be used to determine both the destination node and the packet priority according to our last priority-assignment scheme.

It is interesting to note in the above proof that if we relax the restriction of a static message environment and allow new packets to be continuously injected into our network, the conclusion of the occurrence of packet deliveries remains valid and allows us to establish *livelock freedom* in our network. In other words, the packet-priority ordering defined in equation 2.2 alone suffices to guarantee global progress in a message network operating under a dynamic environment. However, no corresponding statement can be made concerning each individual packet.

2.4.3 Dynamic Environment

The above lemmas give us almost what we want. However, in a dynamic environment where new packets are continuously generated, it is necessary to modify the above priority schemes so that the arguments employed in the proofs of the above lemmas remain valid. As an example, in order to extend the fixed-priority ordering defined in equation 2.1, we need to:

- Assign a unique packet priority to every packet currently in transit inside the network.
- Once a packet has been assigned a particular priority, the message system can only assign higher priorities to at most a bounded finite number of packets during the lifetime of this packet.

The first condition maintains the linear ordering for the priorities among all potential competitors and can be satisfied simply by having the priority values chosen from members of a linear order of infinite cardinality. The second condition replaces the finiteness property of its static counterpart. In particular, an extension, $P = (B, N)$, of the fixed-priority ordering can be defined as follows:

$$(B_1, N_1) > (B_2, N_2) \Longleftrightarrow (B_1 < B_2) \vee ((B_1 = B_2) \wedge (N_1 < N_2)) \qquad (2.3)$$

where B is a packet's *birthdate*. Similarly, to extend the distance-priority ordering described in the preceding discussion, we observe that it has the defect of allowing newly injected packets that have shorter distances from destinations to defeat older packets that have longer distances from destinations. It is not hard to imagine a pathetic case where a packet is defeated indefinitely in channel-access competitions by a steady stream of newly injected packets. This situation can be rectified by assigning lower priorities to the younger packets as follows:

$$(A_1, D_1) > (A_2, D_2) \Longleftrightarrow (A_1 > A_2) \vee ((A_1 = A_2) \wedge (D_1 < D_2)) \qquad (2.4)$$

where A is a packet's *age*, *ie*, the number of routing cycles that have elapsed since the injection of the packet. The age component of this new priority ordering is equivalent to the birthdate component defined in equation 2.3, since all the packets of a network age together. In fact, replacing the birthdates in equation 2.3 with ages will result in a priority ordering identical to the original. We now prove that the priority orderings defined in equations 2.3 and 2.4 guarantee eventual delivery of every individual packet in a dynamic operating environment.

Theorem 2.5 Any packet-to-channel assignment scheme that observes the priority-assignment ordering, as defined in either equation 2.3 or 2.4, during channel-access competitions, guarantees eventual delivery of every individual packet inside a finite network.

Proof. During any routing cycle, let P denote the set of packets currently in transit inside the network, and let $S = \{p_i \in P | A_i \geq A_j \; \forall p_j \in M\}$ denote the set of oldest-age packets. We observe that the packets in S form a static population that satisfies the premises of either Lemma 2.3 or Lemma 2.4; hence, every packet in S will eventually be delivered. As $|S|$ is reduced to 0, packets of the next-oldest age group then become members of S, and the above argument is applied again. With either of the priority orderings defined over a finite network, each packet injected can only have a bounded finite number of packets that are older; therefore, we are guaranteed that every individual packet will be delivered eventually. ∎

The above theorem provides us with well-defined packet-priority schemes that can be employed to assure eventual delivery of every individual packet, and, hence, establishes

lockout freeness in our adaptive networks. There is, however, a practical issue that needs to be settled: In a continuously operating message system, we would like to put an upper bound on the maximum age of any packet ever routed in the network. This is necessary in order to allow for a bounded finite implementation of packet priorities. We now show that such a bound exists as long as packets arriving at their destinations are consumed.

Theorem 2.6 Letting M be a finite multicomputer network that satisfies the consumption assumption and observes the priority orderings defined above, every packet ever routed in the network will then be delivered within a bounded finite period.

Proof. We observe that at the time of injection of each packet inside the network, there can be at most a bounded finite number of older packets already there. Since the highest-priority packet in the finite network will always be delivered within at most DL cycles, where D is the length of the *longest* route defined under $\mathcal{R}$, therefore, every injected packet is guaranteed delivery within a bounded finite period. ■

The above theorems assure us that every packet sent will be delivered within bounded time to its destination under the consumption assumption. There remains, however, the problem of injecting a packet into the routing network in the first place. Notice that for a node located inside a region of heavy network traffic (see Figure 2.7), new packets generated by it have to compete for channel access with packets already in the network. An immediate, naive response would be to simply treat a newly generated packet just like any other packet and have it compete for channel access according to the assigned priority. In the next section, we shall see that the problem is actually deeper than it appears.

2.5 Packet-Injection Guarantees

One major resource limitation that reveals itself when the node *grain size* shrinks is that of limited buffering capability per node. In addition to requiring a steady-state dynamic equilibrium between incoming and outgoing packets, each node also becomes less tolerant of transient fluctuations. In a deterministic blocking scheme, transient fluctuation of message traffic is handled by blocking until the required buffers are released; in our adaptive scheme, this is handled by maintaining a strict balance between incoming and outgoing packets. This is possible only because we are willing to route packets away from their respective destinations if necessary. However, when new packets are injected into the network from a node, this delicate balance is violated locally. The opposite happens when a packet arrives at its destination and is consumed. This violation stems from the unavoidable decoupling of message generation and message consumption in both temporal and spatial dimensions. As message packets are generated at each node and injected into the network, the input/output dynamic balance is violated locally. Consider the situation at the message interface of a node $n \in N$ that has b internal message-packet buffers. Suppose this node is located in the middle of heavy network traffic, so that its input and output channels are continuously busy for a long period. Any message packets generated from the node have to be queued up in extra internal buffers, and compete with the passing message traffic for access to the output channels.

The packet-priority schemes assure that once the queued-up packets have gained access to the network, they will be delivered to their destinations. However, the worst case happens when passing message traffic remains heavy for a prolonged period, and the node runs out of internal buffers after it has injected b packets into the network. Further injection of message packets would then have to wait for internal buffers to free up. Conservation of packets requires that the internal packet buffers remain filled unless:

1. The node receives some input message packets destined to itself, whereby it consumes them and frees up their corresponding buffer spaces.
2. Some input channels become idle for a number of cycles, *ie*, receive *null* flits over those cycles; as a result, the node may free up some of its internal buffer spaces.

In the following subsections, we shall describe two different approaches to assure message injections at every node; each of these will result in guaranteeing the eventual occurrence of one of the above two conditions. But, first, we need to describe in greater detail the possible injection mechanism implemented at the message interface under our adaptive cut-through switching formulation.

2.5.1 Packet-Injection Mechanism

Recall that, in our model, the message interface of a node is the piece of hardware that couples the node's communication subsystem with its computation subsystem. In essence, it is responsible for the checking of destinations of arriving packets, the reassembly of multipacket messages, and the control of packet injection into the adhering communication network. Figure 2.9 depicts a possible conceptual realization of the injection mechanism within the message interface. Its operation is similar to the register insertion ring interface described in [31]. It uses two FIFO buffers that can be connected to the output channel toward the network via a switch. Whenever the node has a packet to transmit, it loads the packet into the injection buffer as soon as the buffer becomes empty. When message traffic arrives from the network input channel, it passes through the destination check logic, which redirects any traffic destined to this node into the node memory. Any remaining passing traffic is loaded into the cut-through buffer, which is normally connected to the output channel. Whenever the cut-through buffer becomes empty, the control logic checks to see if there is an output packet waiting for injection. In such case, the switch is toggled so that the output channel is connected to the injection buffer and the injection proceeds. As the output packet is being forwarded, any passing traffic is loaded into the cut-through buffer. The switch connection is flipped back to the cut-through buffer after injection has finished, and the process repeats. More sophisticated versions of this injection mechanism will use several cut-through buffers in a round-robin fashion; injection of packets can proceed as long as there remains *at least* one empty cut-through buffer at the initiation of injection.

The main interesting property of the message interface for our current discussion is that it provides a mechanism for capturing and accumulating interpacket gaps, which need not be contiguous, as empty spaces inside the cut-through buffers. When enough space has been collected — *ie*, the entire packet length, and, hence, an entire empty buffer — another new packet can be injected into the network. With this mechanism, the question of assuring eventual packet injection is transformed into that of assuring arrival of enough interpacket gaps whenever a node has a packet queued for injection.

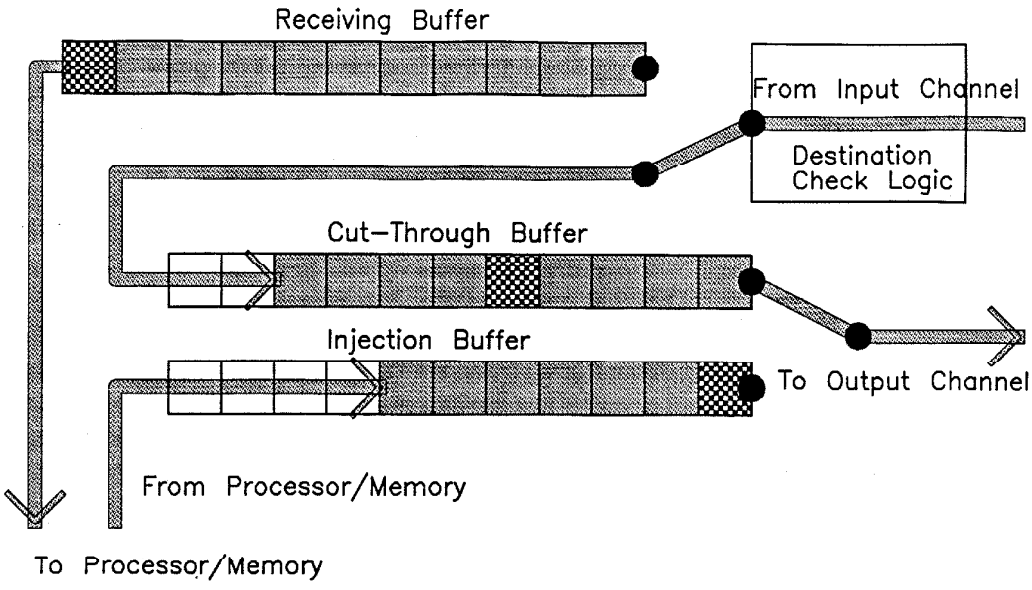

Figure 2.9: Inside the Message Interface

2.5.2 Token-Recirculation Scheme

In this subsection, we describe a method whereby a node can guarantee the emptying of its cut-through buffers inside its message interface by consuming input packets destined to it. In general, a node cannot depend on the other nodes to send message packets to it since it may not be the destination of messages generated by any other node in the network. When a node sends a packet to a different node in the network, it has the effect of migrating a *hole* from the source node's internal buffer to the destination node's internal buffer. In our scheme, we require that the destination node return this hole back to its *owner* by sending a round-trip packet back to the source node. In essence, the internal empty cut-through packet buffers represent reusable tokens that recirculate between the senders and receivers of messages.

Specifically, let b_i be the total number of internal packet buffers in the communication domain inside the message interface of a node $n_i \in N$, and let $0 < h_i \leq b_i$ be a *threshold* that determines the condition under which to trigger the recirculation mechanism. All packets injected into the network are tagged with either a *one-way* or a *round-trip* token. The packet-injection scheme works as follows: Whenever a node is about to inject a packet, it checks its current stock of empty cut-through buffers. If the number is above the specified threshold, the packet to be injected is tagged with a one-way token, otherwise, it is tagged with a round-trip token. At the destination, where the packet is consumed, the carried token determines the receiver's action. The receiver simply consumes the packet if is a one-way token or if it owns the received round-trip token. On the other hand, if the token is a round-trip token belonging to some node other than the receiver, the receiving node is obliged to return it to its owner by sending a reply packet that carries this round-trip token back to its owner. In this

way, the token is recirculated back to its owner by the reply packet, which then consumes the reply packet. This assures the eventual occurrence of packets destined to a node if that node has a packet awaiting injection due to the absence of empty internal buffers. It is straightforward to see that this suffices to guarantee eventual injection for each pending packet. Observe that the validity of our packet-injection guarantee depends on our ability to guarantee eventual deliveries for every injected packet in our network.

The above token-recirculation scheme allows us to assure eventual packet injection at each pending node of the network. However, the scheme suffers from the inherent inefficiency of having to route reply packets in order to return the empty buffers (round-trip tokens) back to their owners. The threshold h specifies the precise condition that triggers this recirculation mechanism. When network traffic is relatively moderate, recirculation tends to be isolated and temporary. As traffic density continues to increase, the population of round-trip packets that carry no useful information also increases. Conceivably, it appears more efficient for a pending node to begin seeking available empty buffers in its own vicinity. Furthermore, large-scale token recirculation commences at precisely that moment when network bandwidth is insufficient. In summary, the token-recirculation scheme leaves much to be desired.

2.5.3 Injection-Synchronization Protocol

The token-recirculation scheme described in the previous section employs a *passive* mechanism whereby each individual node defensively maintains its own ability to inject packets into the network. This is achieved by demanding that packets originating from a busy node be returned to the sender so as to release occupied buffers in the message interface and render the node ready for further packet injections. In contrast, our second approach employs a distributed mechanism whereby each node actively defends its own right to inject packets. The same idea is, in fact, involved in the description of the coherent channel protocol that provides the physical basis of an arbitrarily extensible synchronized network. In particular, when all its buffers are full, the message interface of a node has to wait and collect enough idle time, such as interpacket gaps, from the input channel before it can inject another packet. Our scheme is then a distributed mechanism that guarantees the eventual occurrence of such idle interpacket gaps.

Ideally, whenever a node has a packet queued for injection, it should be allowed to do so. We shall adopt the point of view that a node that has no packet to inject, or is in the process of injecting a real packet, is regarded as if it has a *null* packet ready for injection. Thus, by this convention, ***during every routing cycle, each node has either a null or a real packet ready to inject.*** Our scheme is to introduce *local synchronization* between each node and its neighboring nodes such that the total number of packets injected by a node after each routing cycle cannot differ by more than K, a fixed, positive constant, from those of its neighbors. The null packet convention is required to prevent quiescent nodes that do not have any packet to inject from blocking injections in the active nodes, and *vice versa.*

We now describe the injection-synchronization protocol in a general conceptual framework. For our protocol, we assume that each node explicitly maintains records of the total number of packet injections made by each of its neighbors, measured *relative to that of its own.* For simplicity, the information required to update these records

in each node is assumed to be conveyed by separate direct links between the message interfaces among neighbors during each routing cycle and constitutes an integral part of the network's physical connections. The information exchanged with a node's neighbors during each routing cycle is the total number of packet injections by the node, a 0 or 1, during the current cycle. Records for neighbors' injections are maintained in the form of relative differences, because the absolute value in the number of total packet injections in a continuously operating environment can grow without bound. We now describe the protocol in detail:

0. Immediately after system initialization, each node's records of the relative differences of packet injections for each of its neighbors is initialized to 0.

1. At the beginning of a cycle, each message interface examines the recorded relative differences in the total number of packet injections of each of its neighbors, and computes their minimum. A node is *permitted* to inject its queued packet only if the computed minimum is greater than $-K$, *ie*, if the node is less than K packet injections ahead of its minimum neighbor. Recall that by convention a node always has a packet queued up for injection.

2. If a node is not permitted to inject its queued packet, its message interface just performs its other normal functions and then goes to step 3. If a node is permitted to inject, it examines its next queued packet. Null packets are always injected by convention, whereas real packets are injected only if the injection mechanism described previously finds at least one empty buffer available to absorb the injection transient.

3. Each node computes its own number of packet injections during the current cycle according to the results obtained from step 2, where null packet injections are counted as real injections. This number is then distributed to the message interfaces of its neighboring nodes.

4. At the end of the cycle, each node updates the corresponding record of each of its neighbors by adding the corresponding numbers received from the respective neighbors and subtracting from them the number it computed in step 3. Each node is now ready for the next cycle starting again at step 1.

We now show that following the above stated sequence of actions guarantees that after each routing cycle, the difference in the total number of packet injections between neighboring nodes will be at most K.

Lemma 2.7 The injection-synchronization protocol described above guarantees that after each routing cycle the difference in the total number of packet injections between neighboring nodes will be at most $K > 0$.

Proof. Let c denote the number of routing cycles that have elapsed since initialization. The proof is by induction on c. After $c = 0$, *ie*, immediately after initialization, the differences are exactly 0 in every node, hence, the statement of the lemma is valid. We now assume that the statement is valid after $c = m$, *ie*, the magnitude of the differences I in the total number of packet injections between neighboring nodes is at most K. We now focus on a particular node n. At the beginning of the $(m+1)$st routing cycle, n has

complete knowledge of the differences in packet injections of its neighbors relative to itself up to and including the m-th cycle. By convention, neighbors that are ahead of n in injections have relative differences $I > 0$, while those lagging behind n have as their relative differences $I < 0$. In general, n would have some neighbors having differences zero, some positive, and some negative. If n has some critical neighbors with $I = -K$, then, according to the synchronization protocol, n is denied injection in this cycle. Thus, the change in I for these neighbors of n can only be 0 or $+1$ during this cycle. Therefore, after this cycle, the differences in the total number of packet injections between n and its critical neighbors are still at most k. For those neighbors with $-K < I < K$, no matter what n and these neighbors do in this cycle, their differences are guaranteed to be at most K. For those neighbors with $I = K$, n is itself a critical neighbor of them, and the above argument for the critical neighbors of n can be applied to guarantee a difference of at most K. Hence, after the $(m+1)$st routing cycle, the difference in the total number of packet injections between any pair of neighboring nodes remains at most K. This establishes the validity of the lemma. ∎

The above lemma provides us with a synchronization protocol that guarantees a bounded maximum difference in the total number of packet injections across neighboring nodes in our network. We now show that, with eventual delivery of the packets already injected, this injection-synchronization protocol establishes cooperation among the nodes to assure the eventual occurrence of empty cut-through buffers in the message interface for nodes that have real packets waiting for injection as permitted by the protocol.

Lemma 2.8 A node that has a packet queued for an injection that is permissible under the injection-synchronization protocol will eventually be able to inject, provided the network is livelock free.

Proof. Observe that, by convention, if the pending packet is null, the node is able to inject immediately, so that the lemma is true vacuously. We now proceed to establish its validity for real packets. Suppose, to the contrary, that a particular node, $n \in N$, is blocked from injection indefinitely because the injection mechanism cannot accumulate sufficient empty buffer space to absorb the injection transient. Our injection protocol then dictates that its neighbors also will be blocked indefinitely from injecting. These, in turn, indefinitely block their neighbors, and so on. Given a finite network, all nodes are eventually blocked from any further injection, and eventually *no* new packet can enter the network. Given that the network is livelock free, ultimately it will be void of packets; at that point, the input channel to the interface of n will become idle, thus enabling it to resume the accumulation of empty spaces inside the cut-through buffer. Eventually, it will have collected enough spaces to enable the injection of its queued packet into the network. This contradicts the original indefinite blocking assumption of n, hence establishing the validity of the lemma. ∎

We are now ready to show that by following the above injection protocol every individual node will eventually be *permitted* to inject, and, hence, according to the above lemma, *will eventually inject.* Specifically, let M be a network, and let T_i denote the total number of packet injections from node $n_i \in N$ since initialization. We now prove that T_i is strictly increasing over time.

Theorem 2.9 Given the stated injection-synchronization protocol and a finite network that is livelock free, the total number of packet injections for each node strictly increases over time.

Proof. During a routing cycle, let $t = \min_{n_i \in N} T_i$ denote the minimum among numbers of packet injections since initialization, taken over all the nodes of the network, and let $S = \{n_i \in N | T_i = t\}$ denote the set of nodes that have recorded the minimum number of packet injections since initialization. Since $K > 0$, according to our protocol, every node $n \in S$ is permitted to inject. Lemma 2.8 then guarantees eventual injections from all of the nodes in S; hence, t, the minimum number of packet injections per node, is guaranteed to eventually increase over time. This, in turn, guarantees that T_i strictly increases over time, $\forall n_i \in N$. ∎

Since the total number of packet injections per node is guaranteed to be strictly increasing over time, we are assured of eventual packet injection for each of the individual nodes of the network. In other words, the above theorem establishes *fairness* in network access among all the nodes. We now prove that the above protocol actually allows us to establish *strong fairness* among the nodes of our network under the consumption assumption[1]; *ie*, each node that has a packet pending injection will be allowed to inject within a bounded finite period of time. We now proceed to establish this strong-fairness result.

Corollary 2.10 Letting M be a finite network, as described in Theorem 2.9 above, that satisfies the consumption assumption, then each node that has a packet pending injection will inject within a bounded period.

Proof. We proceed to prove the stated result by exhibiting such a finite, albeit unrealistically pessimistic, bound. Let D denote the network diameter, $\mathcal{N}$ denote the total number nodes, and $\mathcal{L}$ denote a bound on the maximum packet latency whose existence was established in Theorem 2.6 of the last section; and let $t' = \min T_i$ and $T' = \max T_i$ denote the minimum and maximum number of injections per node since initialization among all the nodes of M. Then we have $T' \leq t' + KD$ according to our synchronization protocol. Let t'_i and T'_i denote the respective values of t' and T' after each routing cycle, c_i, $\forall i$; and let c_0 denote the current routing cycle. Since the values of t' and T' increase strictly over time, we are guaranteed that there will exist a routing cycle, c_m, such that $t'_m > T'_0$ and $t'_i \leq T'_0$, $\forall i$, $0 < i < m$. We assert that during the cycles c_1 up to c_m there cannot have been more than $(2KD+1)\mathcal{N}$ packet injections into the network. To see this, observe that we have $T'_m - t'_m \leq KD$ and $T'_0 - t'_0 \leq KD$. Adding them together, we have $T'_m - t'_0 \leq 2kd + t'_m - T'_0$. But c_m was defined such that $t'_m - T'_0 = 1$; hence, $T'_m - t'_0 \leq 2KD + 1$. Since there are a total of $\mathcal{N}$ nodes in the network, the assertion follows directly. Based on Theorem 2.6, we know that there is at least one packet injection within every $\mathcal{L}$ routing cycles. Therefore, the total number of routing cycles, m, elapsed must be $\leq (2KD+1)\mathcal{N}\mathcal{L}$. Since $t'_m > T'_0$, we are guaranteed at least one injection per node in our network every m cycles. The above bound on m then establishes the validity of the required strong fairness result. ∎

[1] As a matter of fact, the token recirculation scheme described in the previous subsection also assures strong fairness to every node.

The above theorems suggest that by following the prescribed injection-synchronization protocol, together with the introduction of packet-delivery guaranteed priorities described in the previous section, we can guarantee progress in every individual node and every individual packet inside a finite network, a very desirable state of affairs.

It is interesting to compare the injection-synchronization protocol with our previous token-recirculation scheme. As has been pointed out, the token-recirculation scheme is inefficient in that a pending node under heavy traffic has to wait for the occurrence of an empty buffer brought back by reply packets across the network. In contrast, in the injection-synchronization scheme, a pending node under heavy traffic prohibits further injections of its neighbors, and the area under *curfew* extends gradually outward if congestion persists. Intuitively, this allows the pending node to capture empty buffers beginning with its local vicinity, which appears to be more efficient. Our null-packet injection convention has the interesting property that an idle node with no packet to inject tends to have its own total number of injections drift toward the maximum among those of its immediate neighbors. As a result, idle nodes tend to exert minimum interference with the activities of busy nodes. Another difference between the injection scheme and the token-recirculation scheme is that under the token-recirculation scheme, more round-trip packets are injected into the network precisely when heavy congestion occurs. This further overloads the network resources and can increase the message latencies substantially. In contrast, the injection scheme tends to regulate total network packet population, which prevents message latencies from growing excessive under heavy-traffic condition. In fact, as we shall see in the next chapter (section 3.5.4), the injection-synchronization protocol can be extended to serve as a *congestion control* scheme to help stabilize the network operating points within regions that deliver favorable performance, regardless of the external applied load. Such schemes indeed provide a *practical* alternative to the *theoretical* priority-based delivery guarantee strategy, which appears to be rather formidable to realize on silicon in its present form.

2.6 Summary

In this chapter, we have focused on addressing a number of feasibility issues that are fundamental to any message communication network that supports reliable concurrent computations. An adaptive cut-through switching model is presented that allows us to discuss a number of fundamental issues, such as communication deadlock, eventual packet delivery, and eventual packet injection.

An argument based on the desire to be able to exploit alternate routes in order to lessen local network congestions is given that led to the adoption of the misrouting discipline. The basic coherent exchange protocol is presented that, together with the misrouting discipline, renders communication deadlock freedom in cut-through switching a non-issue. This, in turn, allows us to exploit arbitrary alternate routes without having to worry about potential communication deadlock, which is difficult and expensive to avoid using more conventional means.

The necessity of having extra buffer space per node is demonstrated with a livelock example; and the buffer requirement, which is necessary and sufficient to avoid such livelocks, has been determined under a set of reasonable buffer operating assumptions. A general packet-priority scheme for constructing packet-delivery guaranteed routing

algorithms for arbitrary network and adaptive control is developed. The priority scheme developed is then shown to be lockout free, and, under the consumption assumption, shown to assure delivery of message packets within a bounded period over a finite network.

The problem of assuring eventual message packet injection at pending nodes is examined in detail. The two conditions under which a pending node is allowed to inject its queued packet are discussed. A detailed description of a conceptual realization of the packet injection mechanism inside the message interface is given. Two different solutions are then presented, each corresponding to a guaranteed eventual occurrence of one of the two desirable conditions. The two solutions are each examined in detail, followed by a discussion of the relative merits of the two approaches.

The main emphasis of this chapter is to lay down a basic foundation for exploring various strategies and extensions of the adaptive framework for message routing in multicomputer networks. We have focused on addressing the fundamental feasibility issues of guaranteeing deadlock freedom, packet delivery, and packet-injection progress, and we did not encounter any insurmountable problem. Rather, the simplicity of these resolution mechanisms gives us hope that adaptive schemes may be found that will improve on the already highly evolved oblivious-routing schemes. In the later chapters, we shall investigate other theoretical and practical aspects of our adaptive routing formulation, namely, the possible performance gain and reliability enhancement issues.

Chapter 3

Performance

In the last chapter, we concentrated on describing the basic adaptive cut-through message-routing model for tightly coupled multicomputers, and the various problems concerning the *feasibility* of the proposed framework. Specifically, we examined such fundamental issues as communication-deadlock freedom, packet-delivery assurance, and packet-injection assurance. We also discussed in great detail such low level issues as the coherent channel protocol that works for arbitrarily extensible networks, and a packet-buffering discipline for fixed-length packets under realistic hardware assumptions. Having demonstrated affirmatively the feasibility of the adaptive approach, we move on, in this chapter, to investigate issues concerning questions of network *performance*.

Specifically, in section 3.1 of this chapter, we define and discuss the principal performance metrics for the multicomputer networks in which we are interested. In particular, we shall focus on the cases of 2D and 3D rectilinear networks, and derive simple theoretical bounds on their various average performance figures. These low-dimension networks are chosen as representative of regular networks that are readily realizable in practice. In section 3.2, we present a qualitative discussion of the operating characteristics and message traffic pattern in multicomputer networks, followed by a discussion of the corresponding implications to performing adaptive-routing decisions. Following these discussions, in section 3.3, we motivate and develop a simple stochastic switching model based on certain simplifying assumptions. We then use the model to derive a number of analytic approximations that provide insights into the nature of our adaptive-routing formulation. In section 3.4, we describe in detail a set of simulation experiments devised to explore tradeoffs and behaviors of the various performance characteristics of our adaptive cut-through routing formulation for the 2D and 3D rectilinear networks. We follow this in section 3.5 with a detailed discussion of the experimental results. Whenever possible, interpretations are given in accordance with the understanding derived from the theoretical model developed in section 3.3. Finally, we summarize the chapter in section 3.6.

3.1 The Performance Metrics

We have motivated the adaptive-routing approach as a technique by which we can more efficiently utilize available network bandwidth by exploiting the existence of multiple routing paths to common destinations that is inherent in the communication networks

that connect multicomputers. The two most important performance figures for routing networks are the average message *latency*, and the average message *throughput*. Our use of the term *average* here presupposes the existence of a *steady-state* traffic over the long term. While, in reality, such an average may never be achieved, its various approximations, nevertheless, provide useful indicators of overall network performance.

3.1.1 The Principal Performance Metrics

In this section, we shall define and discuss the principal performance metrics of multicomputer networks. For our present purpose, we shall make the simplifying assumption that the network operates *synchronously* in discrete routing cycles. The synchronous assumption establishes a direct correspondence between elapsed time and elapsed cycles, allowing us to use the discrete quantity as a convenient measure. Bearing in mind that we are primarily interested in the time average of these performance metrics, here are our definitions:

Definition 3.1 The *channel utilization* is the fraction of time a channel is busy transmitting data. The *injection rate* of a node is the rate at which the node is injecting *new* packet data. In the absence of misrouting through the internal channel, the injection rate is equal to the utilization factor of the internal channel.

Definition 3.2 The *packet latency* is the total number of elapsed cycles from the time the first flit of a packet enters the network at the source-node message interface to the time when the last flit of the packet leaves the network at the destination node message interface.

Definition 3.3 The *message latency* is the total number of elapsed cycles from the time the first flit of a message enters the network at the source-node message interface to the time when the last flit of the message leaves the network at the destination-node message interface. For single-packet messages, message latency is identical to packet latency.

Definition 3.4 The *throughput* of a network is defined to be the total number of message data flits delivered, *ie*, consumed at their destinations, by the network per cycle.

Network throughput, unlike latency, is a performance figure that is independent of the packet and message lengths of the underlying network traffic. It measures the quantity of service provided by a particular network and its routing algorithm. The message latency, on the other hand, is a performance metric that measures the quality of the provided service. Depending on the operating assumptions, the packet latency can be further decomposed into four contributing components:

- *Processing Delay:* The total time spent in computing the output channel assignments at each intermediate node along the routing path. This delay is proportional to the number of hops along the routing path joining the source and destination nodes.

- *Propagation Delay:* The total time elapsed between the time when the first flit of a packet leaves the source node to the time when this flit arrives at the destination node, assuming no queueing at the intermediate nodes. This delay is also proportional to the length of the routing path joining the source and destination nodes.

- *Transmission Delay:* The total time elapsed between the time when the first flit of a packet is received at the destination to the time when its last flit is received. This delay is proportional to the length of the transmitted packet.

- *Queueing Delay:* The total time a packet spends waiting in queues inside the intermediate nodes along the routing path. This delay is a highly nonlinear function of the utilization factors of the communication channels along the routing path.

Because it is clear that the processing delay can be absorbed into the propagation delay, as both are directly proportional to the length of the routing path, this shall be ignored in our subsequent discussion. An ideal routing algorithm should support an average message throughput that is close to the upper limit set by the physical network bandwidth, and have an average message latency that is close to the lower limit set by the average message length and the message distance from destination. However, these two performance metrics are not independent of each other, and are both influenced by other factors as well. For example, under reasonable assumptions, one will be able to increase the maximum sustainable throughput toward the upper limit set by the network's physical bandwidth by adding more internal buffers per node. Similarly, by reducing the amount of message traffic, hence, the network throughput, it is possible to decrease the average latency toward the lower limit determined by the communication patterns. Qualitatively, the effect of having a better routing strategy under heavy applied-load conditions is to realize a more favorable characteristic curve along which the network operates. The major challenge in the design of network routing algorithm is that most state information necessary to arrive at good routing decisions is *distributed* globally over the nodes of the network. Moreover, the information regarding the local states of each node *dynamically* changes over time. Such changes are particularly notorious in the message patterns generated in networks that support fine-grain concurrent computations. These traffic are highly bursty and tend to be transient in nature [3,13].

Another performance figure that we are interested in is the extent to which packets sent between a source-destination pair arrive *out of sequence*. Recall that in our adaptive-routing formulation, packet trajectories are nondeterministic, allowing packet traffic between a pair of source and destination to arrive out of sequence. In general, this puts extra demand on the node memory due to the need to buffer received packets that are awaiting message reassembly and message-order resequencing. Hence, on the average, received packets have to be stored in the node memory for a longer period before they can be processed. The extent to which this happens has a major impact on the storage requirements at each node.

3.1.2 Bounds on Network Performances

Before proceeding to the modeling and analysis of our adaptive-routing formulation, it is natural to first examine in greater detail the performance bounds imposed by the

physical limits. In this section, we shall derive the theoretical bounds on the *average* message latency and network throughput for the general classes of k-ary-n-cubes and n-dimensional meshes. Lower dimensional members of these regular topologies represent networks that are readily realizable in practice. For our present purpose, we assume the following:

1. *Uniform* network traffic pattern; *ie*, each node in the network is equally likely to be the message destination of each generated packet.
2. *Independent* and *homogeneous* network injection rate; *ie*, message packets are independently generated at *identical* rates at each node of the network.
3. *Wormhole* or *virtual cut-through* routing of packets or messages.

Specifically, we shall derive quantitative bounds on the average message latency and the average node-injection rate, based on restrictions imposed by the statistical properties of the uniform traffic pattern and the different network bisection bandwidths [58].

Lower Bound on Average Message Latency

From the previous discussion of the various components of packet latency, it is clear that by ignoring processing and queueing delay, we obtain a lower bound on the message latency as follows:

$$\text{Message Latency} \geq \text{Message Distance to Destination} + \text{Message Length}$$

More importantly, this inequality remains valid for the average values of the above quantities. Assuming uniform network traffic, the average message distance to destination, D_{torus}, for the n-dimensional torus with total number of nodes, $N = k^n$, assuming k even, can be obtained as follows:

$$\begin{aligned} D_{\text{torus}} &= \frac{1}{k}\left(2\sum_{i=1}^{k/2} i - \frac{k}{2}\right) n \\ &= \frac{n}{4}k \end{aligned}$$

where we have taken advantage of the node-symmetry in torus networks, and the statistical independence across different dimensions for uniform traffic in our calculation. Similarly, the corresponding average value, D_{mesh}, of the n-dimensional mesh is given by [3]:

$$\begin{aligned} D_{\text{mesh}} &= \frac{n}{3}\left(k - \frac{1}{k}\right) \\ &\approx \frac{n}{3}k \end{aligned}$$

for any realistic value of k. As an example, for a uniform message traffic on a 32×32 2D mesh with an average message length of 96 flits, we may conclude that the steady-state average message latency must be ≥ 117 cycles, and ≥ 112 cycles if the network is a 2D torus of the same size.

Bounds on Average Network Throughput

We now derive bounds on the average network throughput, or, equivalently, the average node injection rate, imposed by the network bisection bandwidth restriction. Consider an n-dimensional mesh with $N = k^n$ nodes: assuming again that k is even, the network bisection bandwidth is $= \frac{N}{k}$. The bisection can be visualized as a cut by a hyperplane of n–1 dimensions that is orthogonal to the axis of one of the original n dimensions, splitting it into two halves each with $\frac{N}{2}$ nodes. For uniformly random traffic, any message sourced at nodes on one side of the cut will have a 50% chance of being destined for nodes on the other side of the cut. Hence, for q, the average injection rate at each node is:

$$\begin{aligned} q\left(\frac{N}{2}\right)\left(\frac{1}{2}\right) &\leq \frac{N}{k} \\ q &\leq \frac{4}{k} \end{aligned}$$

The above expression gives an upper bound for the average node-injection rate, q, measured in the normalized unit of *flits/cycle*. Notice that the derived bound is independent of the dimension of the network. Similarly, recall that the n-dimensional cube is almost identical to the mesh except that it has *end-around* connections. Hence, its bisection bandwidth is twice that of a mesh of identical dimension. This gives:

$$q \leq \frac{8}{k}$$

For networks of reasonable sizes, *eg*, a 32×32 2D mesh, the network bisection bandwidth limits the node injection rate to $\leq \frac{1}{8}$ for a steady-state uniform traffic pattern, even under the best of circumstances. It is interesting to note that for networks of small radix, the network bisection bandwidth may actually *not* be the communication bottleneck. For example, for the binary 6-cube or, equivalently, the $4 \times 4 \times 4$ 3D torus, the above bound evaluates to $\frac{8}{k=4} = 2$. In other words, as long as the internal channel is of the *same* width as the network channels, the network channels in the 6D cube will never be loaded to more than 50%; rather, in this case, the bottleneck has been shifted to the internal channel.

3.2 Adaptive Cut-Through Switching Decision

Intensive research in packet-switching networks, such as the ARPANET [27,39], has produced many interesting and useful results, and a complete overview of this literature is beyond the scope of the present discussion. Of prime interest to us is the research results concerning practical adaptive-routing algorithms in these loosely coupled networks. Almost all of the proposed adaptive routing algorithms to date have been based on the notion of the *shortest paths* [10] between nodes. In these schemes, the *length* of a communication link between two nodes is assigned a value which is assumed to characterize the amount of congestion along that link. Given a consistent global assignment of congestion values that reflects the current traffic situation, a shortest path between a source and destination would represent a minimum latency route for the message. Adaptability to local network congestion fluctuations is achieved by periodic shortest-path calculations that keep updating the assigned congestion values. However,

such schemes are impractical in tightly-coupled multicomputer networks for at least two important reasons:

1. The highly bursty and transient character of the message pattern generated by concurrent computations demands high-frequency updates of congestion estimates. The overhead required in the collection and periodic broadcasting of these congestion estimates over the network will dramatically reduce the available network bandwidth for normal message traffic.

2. The successful operation of the shortest-path routing algorithm requires each node to maintain global knowledge of the network. Specifically, the shortest distance to every possible destination has to be explicitly maintained in each node in order to accurately reflect the current network condition. Given the scarcity of hardware resources per node in massively concurrent multicomputers, maintenance of global knowledge would be far too expensive, and infeasible in the case of a fine-grain machine.

Therefore, instead of relying on periodic shortest-path calculations to adapt to changing network conditions, and, hence, achieve low-latency routing, one is led to consider practical adaptive control that will perform routing decisions based solely on local information available at the time of decision. The local decision policy adopted may assume a variety of different forms. For example, a dispersive policy was suggested originally in [8] for permutation routing. Under this policy, a node would always send away any incoming message regardless of whether it can make any progress. The rationale suggested is that by sending the competing messages to adjacent nodes, we may manage to enlarge the locally congested bottleneck. The main difficulty with such dispersive approaches is that indiscretionary misrouting can be counterproductive, especially when employing cut-through switching, or when under heavy network traffic. On the other hand, it is generally very difficult and expensive to make discretionary misrouting choices, since any such decisions necessarily involve more than just local information, *eg*, as required to distinguish local congestion from global congestion. Henceforth, we shall adopt a more conservative policy toward misrouting, employing it only as a defensive mechanism to prevent buffer overflow. Instead, we depend on the ability of our multipath control to exploit the rich connectivity inherent in most popular multicomputer networks. This capability to direct messages along many alternate paths tends to smooth out any temporary congestion and disperse the local traffic hotspots. The richer the connectivities, the more numerous the number of alternate routes a message can follow, hence, the more effective this dispersive strategy will be. ***Adaptation*** occurs entirely at the local level where messages are forwarded to one of their profitable channels depending on the specific traffic assignments at the time of decision.

Exactly how each packet gets assigned to which output channel depends on the underlying network topology and the actual mechanics of the local assignment process which may assume a variety of forms. However, they are all expected to obey the following ***no-idle*** policy: Whenever an output channel becomes available and there is at least one packet waiting for the idle channel as one of its profitable channels, then the channel will be assigned to one of these packets within a short delay period depending on the processing requirements. In other words, while it is a well-known fact from scheduling theory that following the no-idle assignment policy could cause anomalies in

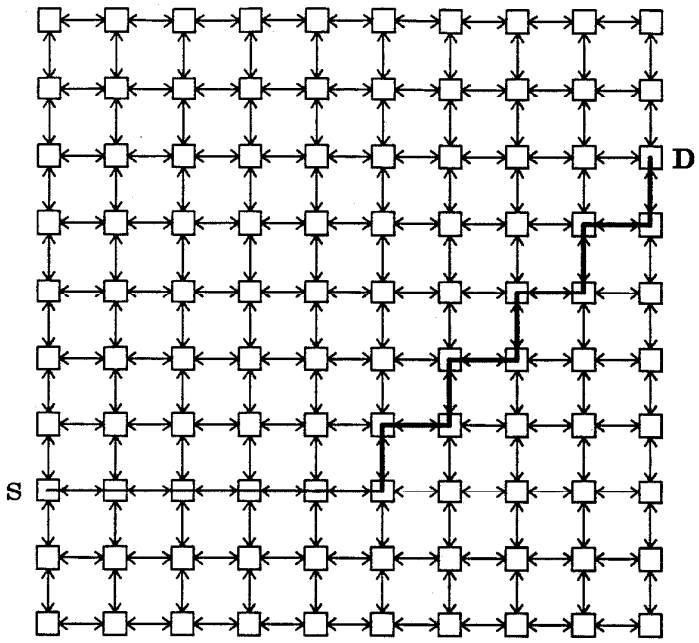

Figure 3.1: An Assignment Decision Having a Preferred Direction

scheduling, concerns over the raw routing logic complexity (as well as speed in general) dictate the use of simple, greedy assignment heuristics.

Another common property likely to be shared by all reasonable assignment logic can be obtained from the following observation: In virtual cut-through switching, packets generally will arrive at and leave the intermediate nodes at arbitrary times. Although each router has a multiple number of input channels, the arrival of a multiple number of packet headers at the *same* cycle tends to be very rare. Similarly, in steady-state operation, it is very rare for a router to assign a multiple number of packets to output channels at the same cycle; in a majority of cases, the routing assignments are made in a *sequential* and *event-driven* fashion. Furthermore, the routing decisions are mostly *single* packet-to-channel assignment decisions. As we shall see in the next section, these properties are sufficient to allow us to derive matching statistics that remain valid over a wide range of possible assignment heuristics.

In the basic, adaptive cut-through routing framework described in the previous chapter, we have formalized the notion of a local multipath routing algorithm $\mathcal{R}$ as a set of routing relations, R_i at node n_i, that generate all profitable channels given any message destinations. In particular, for simplicity, we have tacitly assumed that the various profitable channels are all *indistinguishable* since forwarding the packet along any one of these channels will reduce the message-to-destination distance. In practice, this restriction is unnecessary. In fact, as we shall see in the next chapter, the underlying metric employed in the definition of the routing relations may actually suggest certain channels as being more profitable than others. Whether the routing control hardware wants to take advantage of the proposed distinctions will depend mainly on the additional complexity involved.

As an example of the way in which certain output channels may be regarded as more profitable than others, consider the case of a 2D rectilinear mesh, assuming that the routing relations are generated according to the usual L_1 or city-block metric. As we have argued in the previous paragraphs that the ability to route on a multiple number of alternate paths is advantageous, this consideration suggests the following performance enhancement heuristic for the 2D mesh: Wherever there is a choice between two profitable channels, always pick the one that lies along the longer dimension (see Figure 3.1). The rationale behind this preference is that it systematically allows a message to *preserve* the existence of multiple alternate choices for as long a period as it can. Empirical simulation results indicate that for low- to medium-traffic density, the heuristic consistently gives rise to a few-percentage improvement in message latencies. The advantage, however, diminishes when the applied load further increases. Therefore, whether it is in fact desirable to incorporate this heuristic into the adaptive routing control, depends primarily on the amount of extra circuitry required to implement it.

3.3 Stochastic Modeling and Analysis

In this section, we shall first describe a very simple operational model for our adaptive cut-through switching formulation. We then use this model to perform a simplified probabilistic analysis in order to derive the first-order theoretical tradeoff relationship between packet latency and throughput. Our main objective in performing this analysis is to promote understanding of the mechanics of adaptive-routing decisions. We start by first defining a number of symbols to be used in the subsequent development. Some of these symbols have already been used in previous sections, and are repeated here for the sake of completeness.

Definition 3.5 Let p denote the channel *utilization* factor, *ie*, the fraction of time a channel is busy.

Definition 3.6 Let q denote the *injection* channel utilization factor, *ie*, the fraction of time a node is injecting new packets.

Definition 3.7 Let c denote the *degree* of a node, *ie*, the number of bidirectional network channels that join it to its neighbors.

Definition 3.8 Let b denote the total number of packet buffers available at each node.

Definition 3.9 Let d denote the average *distance* a packet has to travel. Notice that for very low traffic density, the average message latency should be approximately equal to the sum of the average packet-distance to destination and the average message length, assuming a metric-based routing relation.

In general, without any *a priori* knowledge of the computations being performed over a multicomputer network, it is impossible to determine the applied traffic load, and, therefore, the network performance. The following set of assumptions, not all independent, are chosen to represent a reasonable generic traffic demand and to allow for a tractable analysis. Again, several of the assumptions have already been used in previous sections, and they are restated in greater detail here.

1. The network is *node symmetric*, *ie*, the network topology appears to be identical from the viewpoint of every node of the network.

2. Network traffic density is *homogeneous* all over the network; in particular, the average channel utilization is identical over all channels of the network.

3. Packets are generated and injected into the network at each node *independent* of each other. The packet-arrival processes at each node are assumed to be time-invariant *Poisson* processes with *identical* injection rates.

4. The message packets are generated to destination *uniformly* over the entire network. The average message distance-to-destination d, is dependent upon the specific network size and topology under consideration.

5. The applied load is within the network bandwidth limit, and the network traffic has settled into a *steady-state* pattern; *ie*, the average node-injection rate and average message-arrival rate are equal.

6. Packets have a *fixed* identical length $= L$ flits. Longer messages are broken into a multiple number of fixed-length packets, with each packet routed independently by the network.

7. Packets are routed in the *virtual cut-through* fashion; *ie*, once an idle output channel has been assigned, the selected packet will be forwarded immediately without waiting for the complete arrival of the entire packet.

8. Complete routing information is contained in the header flit of length *one*; *eg*, we can regard the header as holding the packet destination node address. Under this assumption, the propagation delay per intermediate node is one cycle.

Assumptions (1) to (4) are simplifications that allow us to carry out the analysis on a *node-by-node* basis, rather than over the entire network. Furthermore, these assumptions represent reasonable approximations in popular networks such as the k-ary-n-cube if some form of load balancing is provided for. More importantly, while these assumptions are seldom directly applicable, they represent a realistic approximation to the case of completely *random* object placements and communications. Randomizing strategies are particularly attractive in concurrent computations that are highly irregular and dynamic in nature [3]. Assumption (5) is an idealization chosen to allow us to proceed with meaningful steady-state analysis. Specifically, it means that the message-injection rate per node generated by the Poisson process mentioned in assumption (3) must be smaller than the upper bound on throughput imposed by the network bisection bandwidth. Assumption (6) is chosen to capture the likely engineering tradeoff involved in a practical VLSI implementation of the adaptive scheme. Assumptions (7) and (8) together made explicit our interest in employing the virtual cut-through switching technique to take advantage of the special operating characteristics of the tightly coupled multicomputer networks mentioned in Chapter 1.

3.3.1 The Assignment Statistics

Since our adaptive-routing formulation is motivated by the desire to exploit the existence of multiple routing paths between senders and receivers, we shall start our analysis by investigating assignment statistics concerning the underlying adaptive mechanism, *ie*, the assignments of packets to their respective profitable channels. As we shall see later, these statistics ultimately govern the performance of our routing formulation. Naturally, the statistics themselves will depend on the network topology, the operational characteristics, and the traffic pattern under consideration. For our discussion, we shall concentrate only on the practical cases of 2D and 3D tori connections, and only for traffic following the assumptions enumerated earlier in this section. We shall also assume that each node has a *sufficient* number of packet buffers so that the amount of misrouting is minimal and can be ignored in our subsequent development. The performance figures obtained under this assumption simply provide an upper bound on those achievable under any realistic, finite buffer restriction.

The routing decision for each packet depends on the respective destination of the packet and on the corresponding state of the router at the time of decision. Consider the case of a 2D torus: Each packet routed in a 2D torus may have anywhere from one to four profitable channels. As long as the network traffic is uniformly distributed, and the average message-to-destination distance is not too small, *ie*, the torus is of a reasonable size, an overwhelming majority of the packets will have exactly two profitable channels when they arrive at an intermediate node. A similar observation reveals that most packets routed in a 3D torus will have exactly three profitable channels at an intermediate node. Furthermore, the assumptions on uniform message traffic, node-symmetry, and homogeneity on link utilization imply that the distribution of profitable channels is *isotropic*, *ie*, equally likely in all directions. These observations allow us to simplify the modeling process by adopting the following approximations:

1. For the 2D torus, the profitable channels of each arriving packet at an intermediate node falls exactly into one of four groups: N and E, S and E, N and W, and, finally, S and W; these correspond to each of the four planar quadrants.

2. For the 3D torus, each packet falls into exactly one of eight groups of exactly three profitable channels, these correspond to each of the eight spatial octants.

3. In both cases, the distribution of profitable channels for each packet is *independent* and *uniform* among the different groups with *equal* probabilities, *ie*, $\frac{1}{4}$ for the 2D torus and $\frac{1}{8}$ for the 3D torus.

Having modeled the distribution of profitable channels for the packets, the next job is to model the routing-assignment process. To proceed, we observe that during every period of L routing cycles, each output channel will be able to forward exactly one packet if there is some packet requesting that channel. In other words, as long as we focus on a time resolution equal to L cycles, each output channel will appear to be available for assignment, for every L-cycle interval. Therefore, we shall take advantage of this and define the assignment statistics over a window of L cycles, *ie*, the total number of successful assignments over a period of L cycles. In particular, we seek an answer to the following question: Given that we start with a node initially having i internally stored packets, how many packets will be forwarded profitably within the next L cycles,

2D TORUS	*Number of Packets*									
Matching Size	1	2	3	4	5	6	7	8	9	10
1	1.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
2	0.0000	1.0000	0.1877	0.0470	0.0116	0.0029	0.0007	0.0002	0.0000	0.0000
3	0.0000	0.0000	0.8123	0.5466	0.3083	0.1626	0.0838	0.0428	0.0214	0.0105
4	0.0000	0.0000	0.0000	0.4064	0.6801	0.8345	0.9155	0.9570	0.9786	0.9895

Figure 3.2: Sequential Assignment Probabilities for a 2D Torus

3D TORUS	*Number of Packets*									
Matching Size	1	2	3	4	5	6	7	8	9	10
1	1.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
2	0.0000	1.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
3	0.0000	0.0000	1.0000	0.0696	0.0087	0.0011	0.0002	0.0000	0.0000	0.0000
4	0.0000	0.0000	0.0000	0.9304	0.2678	0.0740	0.0196	0.0051	0.0013	0.0003
5	0.0000	0.0000	0.0000	0.0000	0.7235	0.5623	0.3367	0.1838	0.0952	0.0488
6	0.0000	0.0000	0.0000	0.0000	0.0000	0.3626	0.6435	0.8111	0.9035	0.9509

Figure 3.3: Sequential Assignment Probabilities for a 3D Torus

assuming that j new arrivals will occur during this same L-cycle interval? In other words, we are seeking the probability, $p_{i,j,k}$, for $i \geq 0$ and $0 \leq j, k \leq c$, where k denotes the number of successfully assigned packets.

We now argue for a simple approximation for the assignment probability $p_{i,j,k}$. To proceed, observe that routing decisions in cut-through switching can be made with very little delay; therefore, it is clear that all $i+j$ packets are legitimate candidates for channel assignment during this same L-cycle interval. Furthermore, given our assumptions that the distributions of profitable channels for all packets are identical and independent, all $i+j$ packets are ***indistinguishable*** from each other from a ***probabilistic*** point of view. Since the distribution is assumed to be isotropic, the channels are likewise indistinguishable from each other. From the discussion at the end of the previous section, we know that the routing assignment decisions are carried out primarily in a sequential fashion. Together, they suggest that the actual assignment statistics can be reasonably approximated by those generated under a strictly sequential assignment process over a total of $i + j$ random packets. In other words, we have $p_{i,j,k} \approx m_{i+j,k}$, where $m_{\alpha,\beta}$ denotes the probability of having β profitable packet-to-channel matchings, given initially a total of α packets. Furthermore, the assignments are performed sequentially, matching one channel after another. To obtain the desired statistics, a *Monte Carlo* simulation was performed and the corresponding results for the 2D torus and 3D torus are tabulated in Figures 3.2 and 3.3. Once we have obtained these assignment statistics, we can use them to obtain a stochastic equilibrium solution for our adaptive cut-through routers. As we shall see, the assignment statistics govern the probabilistic state-transition properties of our system.

3.3.2 Stochastic Equilibrium

Having obtained the necessary assignment statistics, we are ready to analyze the adaptive virtual cut-through routing formulation. In general, ***multi-server*** queueing systems are notoriously difficult, and, except for a few special cases, usually defy analysis. In principle, any stochastic system can be modeled as a Markov process by incorporating

complete information in the state description. In practice, however, such is not feasible, and the challenge becomes that of picking an acceptable state representation with *sufficient* information [5]. In the present case, a complete state description would consist of the total number of packets presently stored in the node; the set of profitable output channels for each packet; whether or not each input and output channel is busy, and for how long; and, for systems with packet delivery guarantee, the relative priority of each stored packet. In fact, just keeping track of the profitable channels for packets in the buffers will require a total of $\binom{3^3+6-1}{6} = 906192$ different states, for a 3D mesh network with six bidirectional channels, and six internal buffers. There is clearly little hope of obtaining the stochastic equilibrium solution for such an enormous system; even if we did, we would be overwhelmed by too many insignificant details. In any case, such a complete state description is simply out of the question!

Another source of difficulty in network traffic modeling arises from the inherent dependency between the packet interarrival times and the packet length [26]. These dependencies render the modeling problem intractable to analysis. Therefore, in order to proceed, we shall adopt the following additional simplifying assumptions:

9. Packet arrivals at the various nodes and channels are *independent* of each other.

10. The packet-arrival distributions are memoryless which, in the present case, is modeled as a *Poisson* process with arrival rate $\lambda = cp/L$.

11. After each period of L cycles, a new set of profitable channels is chosen for each packet from the idealized distribution described in the previous subsection.

Assumptions (9) and (10) closely resemble the well-known *Kleinrock Independence Assumption* [26], and were chosen with the same purpose and motivation in mind. In the original statement of the independence assumption, in addition to memoryless arrivals, each time a packet is received at a node, a new length is chosen from an exponential distribution. Since we assume fixed-length packets, such is not necessary; instead, we replace it with assumption (11), which eliminates the history dependencies in successive rounds of routing decisions. These assumptions, together with our earlier symmetry assumptions, allow us to completely characterize the stochastic state of a node by the total number of packets that are stored internally, *ie*, the *queueing population*. In terms of standard shorthand notation in queueing theory [27], we have something akin to the $M/D/c$ queueing system, *ie*, memoryless arrivals; deterministic service time $= L$, and $c \geq 1$ servers. However, there is one important difference between our case and those studied in conventional queueing systems: *non-identical servers*. This is because each packet has its own choice of profitable channels, hence, a server, *ie*, a channel, may remain idle even if the queue is nonempty.

We are now ready to model the transition behavior of the system. Based on the observation made earlier, we shall observe the state of our system, *ie*, the queueing population of a node, only at the sequence of snapshots at cycles: $0, L, 2L, 3L, \cdots$, *etc.* State transitions occur as a net result of two continuous processes: The arrivals of new packets at the intermediate node, and the departure of packets forwarded by the node. The stochastic equilibrium equations can now be obtained by coupling together the governing statistics of these two processes: the packet-assignment statistics, and

the average packet-arrival rate.[1] In particular, let $Q_i(t)$ denote the state probability of having i queued packets at cycle t. We have, $\forall i \geq 0$:

$$Q_{i+j-k}(t+L) = \sum_{j=0}^{\infty} \sum_{k=0}^{c} Q_i(t) \left[e^{-\lambda L} \frac{(\lambda L)^j}{j!} m_{i+j,k} \right] \qquad \text{where } i+j-k \geq 0,$$

Next, in these equations, we let $t \rightarrow \infty$. Under our steady-state assumptions, $Q_i(t) \rightarrow Q_i$, the equilibrium-state probability, or *time average,* of having i queued packets at a node, and given the ergodicity of the system, this is also equal to the *ensemble average*. Hence, we have:

$$Q_{i+j-k} = \sum_{j=0}^{\infty} \sum_{k=0}^{c} Q_i e^{-\lambda L} \frac{(\lambda L)^j}{j!} m_{i+j,k} \qquad \text{where } i+j-k \geq 0,$$

and the normalizing equation:

$$\sum_{i=0}^{\infty} Q_i = 1$$

Given the matching probabilities $m_{i,k}$, and the arrival rate, λ, these equations can be solved numerically using such well-known successive relaxation techniques as the *Guass-Seidel* method. To continue our analysis, we observe that having the equilibrium state probabilities, Q_i, allows us to determine δ, the expected decrease in the total packet-to-destination distance per node over a L-cycle interval:

$$\delta = \sum_{i=0}^{\infty} \sum_{j=0}^{\infty} \sum_{k=0}^{c} Q_i \left[e^{-\lambda L} \frac{(\lambda L)^j}{j!} m_{i+j,k} \right] k \qquad \text{where } i+j-k \geq 0,$$

which should be $\equiv \lambda L \equiv cp$, the average throughput of a node, as we have ignored any misrouting in our model. This identity serves as an effective accuracy check for any numerical approximation obtained using finite truncation.

Given our assumption of homogeneous traffic, the total expected decrease in packet distance of the entire network over a L-cycle interval is: $\Delta = N\delta$, where N is the number of nodes in the network. The following observation then allows us to determine the steady-state packet-injection rate per node: Since the network is in a steady-state equilibrium, any decrease in packet distance must be balanced by an equivalent average influx of newly injected packets. Therefore, we have $\Delta = NqdL$; after rearranging, we have the average throughput per node: $q = \frac{cp}{dL} = \frac{\lambda}{d}$. We are now ready to pursue our final objective in this analysis, *ie*, to derive an approximate theoretical relationship between packet latency and the applied load. To proceed, we observe that the expected queueing population per node, Q, is given by:

$$Q = \sum_{i=1}^{\infty} i Q_i$$

[1] Observe that under our assumption, it is possible, if not probable, to have more than c packets arriving in $\leq L$ cycles. The independence assumption becomes more accurate with a larger number of channels per node. Therefore, we expect the theoretical predictions for the 3D torus to be better than those for the 2D torus.

On the other hand, according to Little's Theorem [30], we have:

Average Queueing Population = Average Throughput × Average Queueing Delay

From these, we obtain the average queueing delay, $T_q = \frac{d}{\lambda}Q$. Finally, we have for the total average network packet latency, $T_N = T_q + T_p + T_t$, *ie*, expressed as the sum of queueing delay, propagation delay, and transmission delay:

$$T_N = \frac{d}{\lambda}Q + d + L - 1 \tag{3.1}$$

Equation 3.1 allows us to determine the desired theoretical relationship between latency and throughput. Figures 3.4 and 3.5 depict the latency curves obtained for the 2D torus and 3D torus. In these figures, the applied load has been normalized with respect to the upper bound imposed by the network bisection capacity. Similarly, the average packet latency has been normalized with respect to the lower bound imposed by the sum of the packet length and the average message distance to destination. It is clear from the figures that at a very low applied load, the network latency is almost completely incurred by the sum of propagation delay and transmission delay. As applied load increases, the queueing delay starts to dominate and the overall latency approaches values comparable to those obtained in store-and-forward routing [25].

In the foregoing analysis, we have deliberately ignored the traffic through the internal channel connected to the message interface of a node. This traffic represents packets that are injected by and delivered to the message interface of the node. By deliberately omitting this traffic, we obtain results that are intrinsic only to the network and to the routing strategy. To see why, recall that in homogeneous steady-state equilibrium, the additional injected traffic from the message interface is *exactly* canceled by the arriving traffic destined to this node, on the average. Hence, the average queueing population, the average resulting queueing delay, and, T_N, the average *network* packet latency, all remain unchanged.

However, the internal channel does introduce two additional queues into the system: a source injection queue at the message interface of the sender and a destination arrival queue at the router side of the receiver. For a reasonable-sized network, their effects can be safely ignored. Consider a typical 2D torus of size 32×32. The bisection bandwidth limits the injection rate to be ≤ 0.25. In other words, the two additional queues are each operated at a utilization factor of ≤ 0.25. In almost all queueing disciplines, this appears to be a very small perturbation that does not affect the first-order results. Furthermore, it is clear from the above argument that this perturbation decreases as the network size increases. For smaller networks that are shorter in each dimension, however, the effect of this internal channel on the message interface cannot be ignored. Following our assumptions, packets having arrived at their respective destination nodes that are waiting to be sent to the message interface will form a $M/D/1$ queue connected in tandem at the receiver side. For an $8 \times 8 \times 8$ 3D torus network, the injection-channel utilization factor, q, is limited to ≤ 1; therefore, the extra delay can no longer be neglected. The average value of this extra delay, W, is given by the *Pollaczek-Khinchin Formula* [27] for $M/D/1$ queues:

$$W = \frac{q}{2(1-q)}L$$

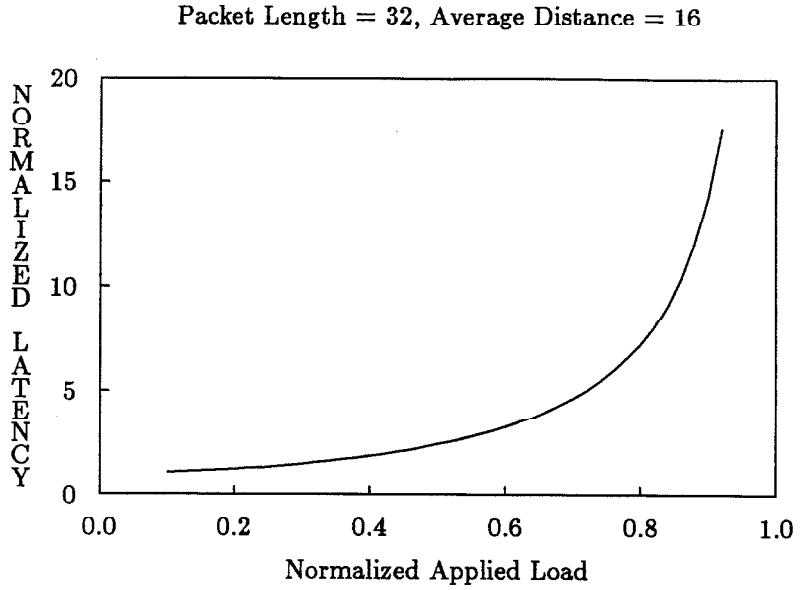

Figure 3.4: Network Latency *versus* Applied Load: 2D Torus

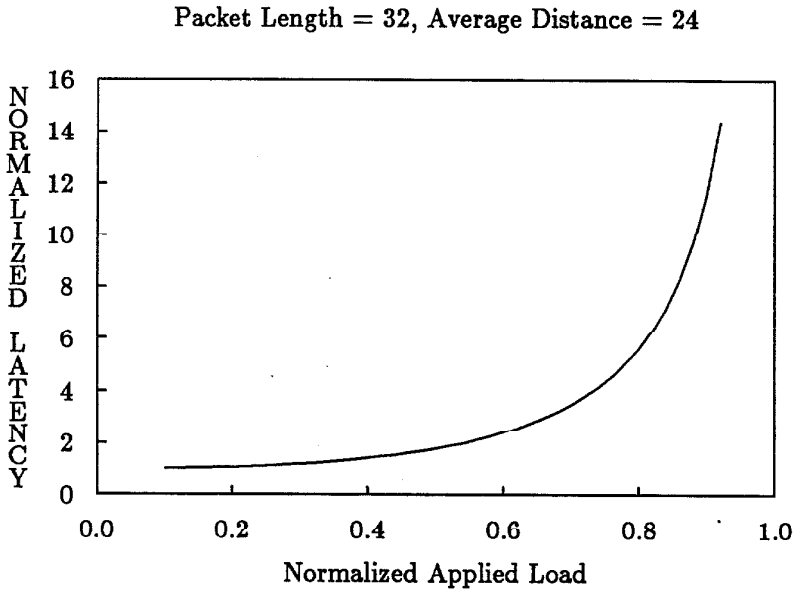

Figure 3.5: Network Latency *versus* Applied Load: 3D Torus

Since, under our assumption, the individual packets are also generated according to a Poisson distribution, we can finally observe that the same $M/D/1$ queue is also formed at the message-interface side for packets waiting to be injected. In other words, the average *source queueing time* is also given by the above formula. We shall have more to say about these extra queues in section 3.5.

3.4 The Simulation Experiments

Our stochastic analyses presented in the previous section were carried out under many simplifying assumptions. These analyses are primarily useful for understanding the qualitative behavior of the networks under our adaptive-routing formulation. For example, modeling the dynamics of a routing node in detail points out the significance of the packet-to-channel matching distribution. This suggests that random object-placement strategies are very attractive in terms of both simplicity and excellent matching characteristics. To evaluate the conclusions derived from our stochastic models, and to gain additional insights into situations that do not satisfy our simplifying assumptions, we performed a set of discrete event simulation experiments employing adaptive cut-through routing on several representative networks. In performing these simulations, we specifically sought answers or insights into the following issues:

1. The accuracy of our stochastic analysis in predicting the network performance under situations that satisfy our simplifying assumptions. Of primary interest here are the general characteristics of performance behaviors. Another interest is the effectiveness of adaptive multipath routing in diffusing local congestions generated in unbalanced traffic, as compared with the performance obtained from oblivious wormhole routing.

2. The network performance for sending and receiving multipacket messages. Since messages generated by computing objects are rarely of the same length as the chosen packet length, longer messages will have to be broken into a multiple number of packets.

3. The interaction between processing delay and communication delay and its effect on overall computing performance. In a majority of concurrent applications, communication traffic is highly *reactive* in the sense that the sending of a new message is predicated upon the receipt of a previously sent message. The result is a kind of negative feedback system that is governed by the highly nonlinear relationship between message throughput, message latency, and processing speed.

4. The effectiveness of performing *congestion control*, based on an extension of the network-access fairness guarantee scheme. Our motive here is to investigate practical schemes that will maintain favorable network performance without incurring substantial overhead. Of primary interests here are the self-stabilizing effects on network operating points, and the second-order performance metrics, *eg*, the standard deviations in messages latencies.

It is clear that, in addition to the ones listed above, there are many other performance characteristics that are both interesting and informative. Rather than attempting a

comprehensive coverage of all these areas, we have chosen instead to focus our investigations on those that we believe are the most important and fundamental, given the realistic constraints imposed by our computing resources.

3.4.1 The Assumptions

We now describe the set of basic assumptions that are common to all the simulations we have performed, and discuss their roles in our simulations. In the subsequent description, we shall use the *flit* as our basic data unit, and the routing cycle as our basic time unit.

1. *Network Topology*: We have chosen to perform simulations on 2D and 3D meshes and tori networks; this gives us a total of four different topologies. These represent regular networks that are practically realizable. The tori, being node- and arc-symmetric, are ideal candidates for investigating the performance of an adaptive scheme in *balanced* traffic. They provide an excellent approximation to our simplifying assumptions. The meshes, on the other hand, tend to produce congestion across nodes located at the center; thus, they are natural candidates for investigating routing under *unbalanced* traffic. All our simulations assume a network with synchronous communication channels.

2. *Packet Format*: For the adaptive router, messages are broken into, and routed in the form of fixed-size packets whose length = 32 flits. For example, the fixed packet size for a typical flit width of 8 bits would be 256 bits. The first flit of a packet is the *header* of the packet, and contains the *address* of the message destination. Being primarily a circuit-switching technique, the oblivious wormhole routers do not break messages into packets; rather, messages are routed as single entities. In both cases, the first flit of the packet or message holds enough information to perform local routing decisions.

3. *Routing Strategy*: Packets are forwarded by the adaptive router in virtual cut-through fashion. Since the headers are only one flit long, packets or messages can be forwarded along their routes without delay at each cycle. For simulations of the adaptive cut-through routing, conflicts in channel access are resolved in two different ways, depending on the simulation objectives:

   (a) *First-come first-served*: where assignments, profitable or otherwise, are performed in a FCFS manner. This simple policy is used only in the congestion-controlled traffic experiments, and represents a realistic scheme that is *practically* realizable.

   (b) *Distance-priority*: where conflicts are resolved according to the packets' distance priorities. This represents a compromise between the desire to guarantee packet delivery theoretically, and to satisfy the information constraint imposed by using *single-flit* headers. This scheme is used in all other traffic experiments.

Packets denied immediate access to channels are stored in buffers inside the adaptive router. Packet misrouting is triggered only if there is danger of buffer overflow.

4. *Buffer Structure*: The buffer storage inside each adaptive router is organized as a collection of 32-flit long FIFO buffers, where leaving and arriving packets can share the same buffer. Each node has fifteen buffers, so as to minimize preemptions and thus obtain results that are indicative of achievable upper-bound performance.

5. *Message Destination Selection*: Message destinations are selected *uniformly* over the entire network, due to the absence of *a priori* knowledge on the type of computations supported. Uniform message destination traffic patterns represent a bound on the network performance that is readily implementable through random placement techniques and offer excellent results in load balancing [3]. See also [53] for an interesting discussion of the process placement problem.

6. *Message-Length Distribution*: For simulations regarding multipacket messages, message lengths are generated according to the *Erlang* (E_k) Distribution; this is a special case of the more general class of Gamma Distribution [57]. The probability density is given by:

$$f(l) = \lambda^k \frac{l^{k-1}}{(k-1)!} e^{-\lambda l}$$

and the cumulative distribution is given by:

$$F(l) = 1 - \sum_{j=0}^{k-1} e^{-\lambda l} \frac{(\lambda l)^j}{j!}$$

for $l \geq 0$ with a mean $E(l) = \frac{k}{\lambda}$ and a standard deviation $\sigma(l) = \frac{1}{\sqrt{k}} E(l)$. For our simulation, we have chosen a mean of 96 flits and a standard deviation of 32 flits. Figure 3.6 shows the selected distribution. The specific density was picked so as to approximate the message-length distribution generated in typical concurrent computing applications [7,29].

The parameter space over which the networks were simulated was chosen under the guidance of our theoretical models and is expected to represent regions of practical interest. We have chosen to simulate networks of medium sizes: $16 \times 16 = 256$ nodes in the case of 2D networks, and $8 \times 8 \times 8 = 512$ nodes in the case of 3D networks. The simulated network sizes were chosen as a compromise between our interest in the performance of the adaptive-routing approach in large networks, and the practical limit imposed by the available computing resources. The enormous computing costs and address space requirements dictate that the simulations themselves be performed on existing concurrent computers. The simulation program is written in C and runs under the *Reactive Kernel* developed here in the Computer Science Department at Caltech [52,55]. The simulation experiments were run almost continuously on the iPSC/1 d7 cube, and later on the iPSC/2 d4 cube and Symult Series 2010 machine [23,24,51]. The entire set of experiments, including a few more from the next chapter, took approximately four months to complete.

3.4.2 The Experiments

We now describe in detail the various sets of simulation experiments we have performed, and the additional assumptions relevant to the individual experiments. The simulation experiments can be conveniently divided into four different categories:

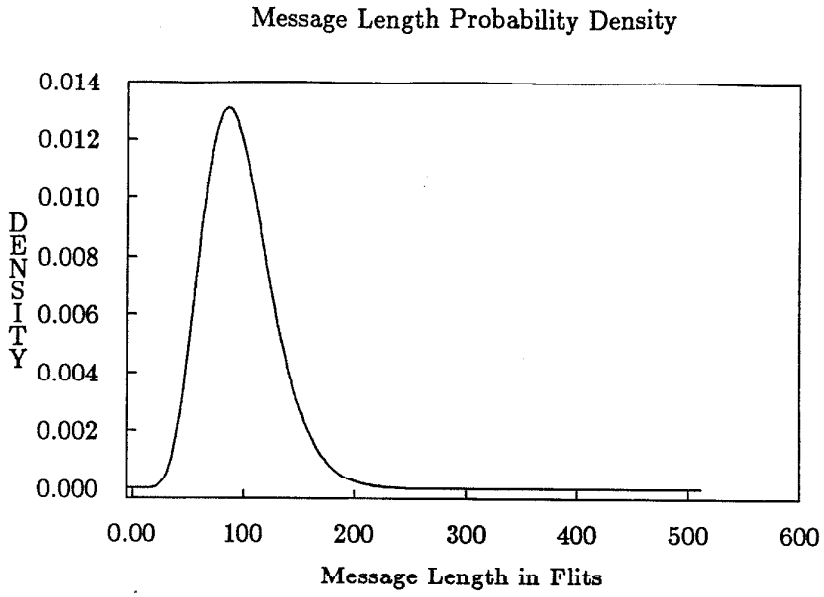

Figure 3.6: Erlangian Distribution: Mean = 96 and Standard Deviation = 32

1. *Independent, Constant Applied Load*: In this set of experiments, messages are generated independently and at a rate that is identical at each node. The message-generation distribution is *memoryless*, which in our discrete case is generated by a *Bernoulli* process of fixed rate. The rate or *applied load* was the parameter that varied in the simulations. These experiments were designed to test the intrinsic network performance figures, and can be further divided into two subcategories:

   - *Balanced Traffic*: The first set of experiments simulated uniformly random message traffic on the 2D and 3D tori. The node- and arc-symmetries in these networks give rise to traffic that is homogeneous everywhere.
   - *Unbalanced Traffic*: In order to test the effectiveness of adaptive routing in exploiting multiple alternate routes around regions of local congestion, simulations of uniformly random message traffic on the 2D and 3D mesh were conducted. The topology of the mesh networks creates highly unbalanced traffic with congestion over nodes around the center of the mesh, thereby providing a testbed for our present purpose. A corresponding set of simulations on the oblivious routers was also performed for comparison.

   Within each category, the experiments can be further subdivided into:

   (a) *Single-packet messages*: These experiments were designed to test the theoretical predictions, and provide indications of performance levels for networks that route relatively short messages.

   (b) *Multiple-packet Messages*: These experiments were designed to test the more realistic case of variable-length messages. Reassembly-buffer population and packet-order preservation are also of interest.

2. *Reactive Message Traffic*: The previous set of simulations were targeted to extract information intrinsic to the communication networks and their routing strategies. In this regard, messages are generated at a constant rate that lies within the limit imposed by the network bisection bandwidth. Our next set of simulations will investigate the impact of interactions between processing delay and communication delay on the overall performance of computing. In particular, in most concurrent applications, computations proceed in a reactive style, in which a process or object receives a message sent to it, performs some local computations, and then sends new messages off to other objects that carry on the computation. In general, the message-generation rate is *not* independent of the message traffic. Rather, it is regulated by the sustained network throughput; this forms a negative feedback system that seeks its own equilibrium. For this set of experiments, each node in the network was endowed with a fixed number of messages at the beginning of the simulation that are ready to be sent. Additional messages were generated at each node upon receipt of previously generated messages sent from another node. The processing delay was scaled in proportion to the length of the received message. In a crude sense, it was an attempt to model the interaction between processing and communication during actual concurrent computations. By varying the scaling factor in the simulations, we attempted to cover a wide spectrum that ranged from processing-intensive to communication-intensive computations. Only multipacket messages were simulated in this set of experiments.

3. *Congestion-Controlled Message Traffic*: In the first two sets of experiments, the packets are allowed to be injected into the network in their earliest possible time, where network congestions have *not* been actively taken into account. In our third set of experiments, we explore the effects of adding congestion control to help stabilize and confine the network operating point to stay within the favorable regions, *ie*, the throughput regions that deliver acceptable network latency values, regardless of the external applied load. In particular, we extended the injection-synchronization protocol described in section 2.5.3, to include additional injection restriction whenever *misrouting* occurs. In other words, as long as a node is misrouting some of its packets, it will not advance its injection count, regardless of whether it has a packet to inject or not. When the misrouted packets have completely left the node, it will revert back to follow the normal injection-synchronization protocol. The rationale behind this congestion-control scheme is that the occurrence of misrouting is symptomatic and, hence, indicative of the *onset* of network congestion. By directly monitoring misroutings, the network packet population can be controlled to stay within regions where misroutings are rare and transient events and, hence, will help to stabilize the network operating point. From this perspective, it is similar to the reactive traffic experiments in that it forms a negative feedback system that seeks its own equilibrium. For this set of experiments, the maximum injection count difference, K, used in the protocol is set to one. Only the mesh networks which inherently create congestions in the center were simulated in this set of experiments.

4. *Fast Fourier Transform Traffic*: In the first three sets of experiments, the traffic patterns were all generated artificially. In our final set of experiments, we chose

to simulate the traffic pattern generated by an actual concurrent computation: the Fast Fourier Transform, on a 16×16 2D mesh. These experiments also gave us an opportunity to compare the effects of different object placement strategies on the overall computational performance. There are many different concurrent formulations for the FFT [50], each corresponding to a different tradeoff point along the $Area \times Time^2$ complexity curve [58]. In our experiment, we chose the fine-grain iterative formulation that gives $O(\log N)$ run time, using $O(N)$ number of nodes. In particular, we performed a 4096-points FFT computation ***in situ***; *ie*, we started with a vector of 4096 data figures, already located in the multicomputer, with 16 at each node. The end result of the FFT computation is another array, *ie*, the transform, that is also spread out among all the nodes of the multicomputer. Since the FFT butterfly operations are distributed over the underlying physical network, the required data distribution has to be implemented by message communication. The particular mapping chosen, however, can affect actual computation performance. In particular, we chose to compare the effects of the two simplest placement alternatives:

- *Systematic Placement*: The mapping used here is simply the *identity* map between logical addresses and physical addresses of the data elements.
- *Randomized Placement*: The mapping used here is a ***pseudo-random bijective*** address mapping that disperses the message traffic.

In addition to routing the traffic generated by the FFT computation, the speed of floating-point arithmetic, or MFLOPS, is another primary parameter that is varied in this experiment.

During each simulation session, except for the FFT experiment, the networks were simulated for total of 80000 cycles which was partitioned into four nonoverlapping intervals of 20000 cycles each. The statistics for the intervals in each simulation were compared and were found to be typically within 2% of each other. This suggests that the simulation interval is long enough for the statistics that were collected. Within each window period, each node typically would have injected and received several hundred messages, and various statistical measures were taken. We now list the set of primary statistical measures taken during the simulations.

1. *Average Message Latency*: The mean time between the injection of the first packet of the message at the source, and the arrival of the last packet at the destination. This is our primary performance metric.

2. *Average Packet Latency*: The measurement of the routing service without the influence of message length or extra reassembly delay. It is essentially a performance measurement of the network.

3. *Average Sustained Network Throughput*: The mean sustained message data delivery rate. Under steady-state condition, it should always be identical to the mean message-injection rate.

4. *Average Source Queueing Time*: The mean time between the generation of a message and its subsequent injection into the network. The total message delay

between generation at source and consumption at destination is the sum of the queueing time plus the message latency.

5. *Average Router Queueing Population*: The mean number of packets queued inside the adaptive router. This gives a rough indication of the number of internal buffers per router required to sustain the desired network performance.

6. *Average Fraction of Out-of-Sequence Message Arrivals*: This measure provides a useful indication of the extent to which messages arrive out of sequence. These messages require resequencing when message-order preservation between sender and receivers is desired.

7. *Average Reassembly/Resequencing Buffer Population*: The mean number of packets buffered at the receiver side waiting for message reassembly and message-order resequencing.

3.5 The Simulation Results

In this section, we present and discuss the results obtained from the simulation experiments described in the previous section. Our emphasis here is the understanding of network performance behavior over the entire range of applied load. Whenever possible, a qualitative interpretation will be given in accordance with our understanding of the adaptive router obtained from the theoretical model.

To facilitate the comparison of experimental results with theoretical limits, we have normalized both the applied load, *ie*, the rate at which messages are generated, and the network throughput, *ie*, the rate at which messages are delivered, with respect to the upper bound imposed by the network bisection bandwidth. As a concrete example, a 16×16 2D mesh network has a maximum throughput of 0.25 flits/cycle for uniformly random traffic. Hence, a normalized applied load of 0.4 indicates that, on the average, a node is generating 1 flit every 10 cycles. As a result of this normalization, when we compare normalized throughput curves of different network topologies, such as torus and mesh, the traffic volume injected and delivered by the torus is twice that injected and delivered by the mesh at the same normalized applied load. Under steady-state conditions, average applied load should be identical to average network throughput. When the applied load exceeds the capacity that can be handled by the network and its routing strategy, the source injection queue size increases without bound.

3.5.1 Single-Packet Messages

The relevant statistics for simulation runs of single-packet message traffic are shown in Figures 3.7 to 3.14. Figures 3.7 and 3.8 plot the latency versus throughput tradeoff curves for the 16×16 2D torus and the $8 \times 8 \times 8$ 3D torus. Also shown in these two figures are the theoretically predicted curves obtained from our theoretical model. Starting at $\approx$ 10% throughput, the latencies in both networks are very close to their theoretical lower bound, *ie*, $32+2\left(\frac{16}{4}\right) \approx 40$ cycles for the 2D torus and $32+3\left(\frac{8}{4}\right) \approx 38$ cycles for the 3D torus. Both curves stay relatively flat until the throughput increases to $\approx$ 70%, at which point the latencies start to climb rapidly and approach values comparable to

those expected in store-and-forward switching. *This phenomenon is characteristic of the virtual cut-through switching technique: The message latency is close to that of circuit switching at low traffic density, and approaches that of store-and-forward switching at high traffic density* [25]. As we shall see later, one of the main differences between the oblivious wormhole scheme and our adaptive cut-through scheme would be in the location of the transition points. Observe that the matching between the theoretical and experimental curves is closer for the 3D torus than between those of the corresponding 2D torus. We conjecture that this is due to the increased number of channels per node in the case of the 3D torus, which makes the *Poisson* arrival assumption more accurate.

Figures 3.9 and 3.10 plot the corresponding latency-versus-throughput tradeoff curves for the 2D and 3D mesh networks. Shown in these figures are the tradeoff curves for both the oblivious wormhole and for our adaptive cut-through results. Oblivious wormhole routing differs from cut-through switching in that it treats the routing paths as data pipelines that join the source and destination nodes, allowing messages to ripple through. Messages trying to access channels currently being used are blocked, which in turn, block other messages behind them in the pipelines [11]. Again, the characteristic shape mentioned above for the virtual cut-through switching tradeoff curves is obtained. For oblivious wormhole routing, the transition points lie at $\approx$ 30 to 40% of normalized throughput, with the maxima never exceeding 50% and 40% for the 2D and 3D mesh, respectively. Figures 3.11 and 3.12 plot the relationships between sustained throughput and applied load for all the relevant networks and routing schemes. Observe that for the oblivious schemes, the sustained throughputs remain stable at their respective maxima even after the applied load exceeds the capacity that can be handled. In addition, the network latencies also remain stable at their respective maximum values. Apparently, the blocking that occurs at those traffic densities is also sufficient to throttle further congestions created by excessive injection. On the other hand, the figures indicate that our adaptive cut-through switching can sustain as much as 85% normalized throughput, given a total 15 packet buffers per node, which is the case in our simulations. Again, the transition points in the latency curves occur at about 70% normalized throughput.

While the average network latencies of the oblivious wormhole routing remain stable for applied loads that exceed their sustained throughput capacities, the corresponding source queueing times increase without bound. Figures 3.13 and 3.14 plot the relationships between source queueing time and applied load for all the relevant networks and routing schemes. Observe that under adaptive routing, the torus network curves resemble those predicted by the Pollaczek-Khinchin Formula for the $M/D/1$ queues. These facts are consistent with the memoryless message-generation distribution used in our simulation experiments. Furthermore, we observe that the curve of the 3D torus is much steeper than that of the 2D torus. That such should be the case can be understood if we recall that the maximum steady-state injection rate of the 3D torus is identical to 1, whereas it is $\frac{1}{2}$ for the 2D torus. Hence, for the 2D torus, the average utilization of the internal channel never exceeds one-half of its capacity. Another interesting point to observe in these two figures is that the average source queueing time for the mesh networks is *lower* than for the corresponding torus networks for all normalized applied loads up to $\approx$ 75%. This may seem like a contradiction at first, as we expect the congestion created at the center of the mesh to result in a much higher average source queueing time than that obtained from the balanced traffic generated over the torus. However,

only those nodes at the center of mesh experience this congestion, whereas the majority of nodes at the periphery are operating with their channel utilizations much less than 1. Hence, there is little or no interference to the traffic carried by the internal channel at these nodes, which are injecting at a rate that is half of the corresponding normalized value for the torus. It is only after the congestion area begins to grow in response to increases in traffic density that the overall average source queueing time begins to climb rapidly and far exceeds that for the torus networks. Hence, in this regard, if a mesh is expected to operate under very heavy traffic density, fairness-guarantee schemes (such as those presented in Chapter 2) should be employed. However, in such cases, because the figures indicate that the torus connections are superior in terms of flow control and fair access to the network, the torus may prove to be a much better topology than the mesh. On the other hand, we might adopt the point of view that a network should never be driven to support a traffic density beyond that of the transition point in its respective latency curve, in which case, the torus has no clear advantage over the mesh. In fact, under a constant bisection capacity assumption, the channel width of a mesh is twice that of a torus, which more than compensates for the longer average message distance to travel.

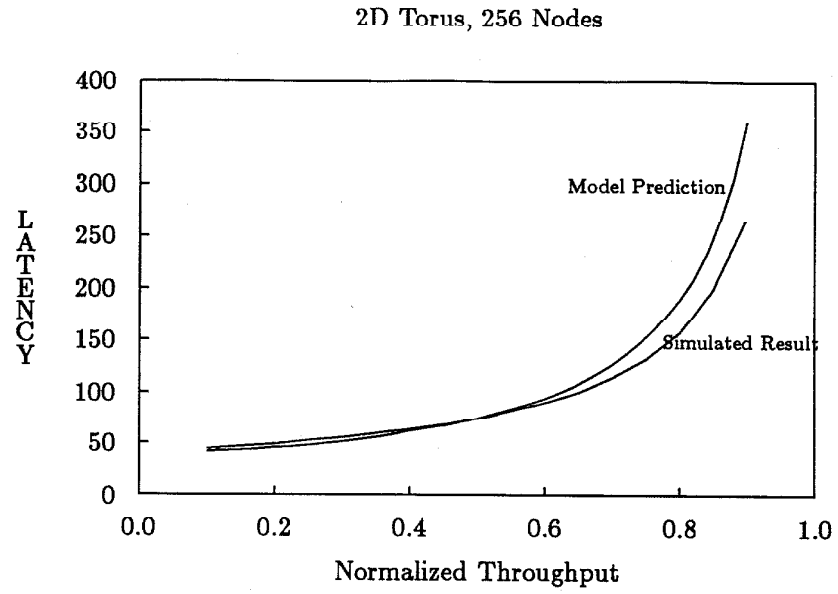

Figure 3.7: Single-Packet Message Latency of 2D Torus

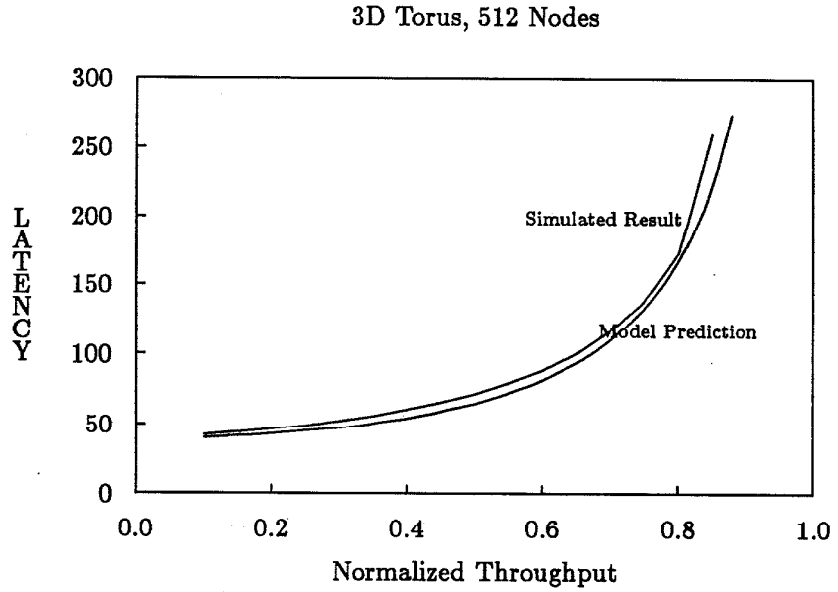

Figure 3.8: Single-Packet Message Latency of 3D Torus

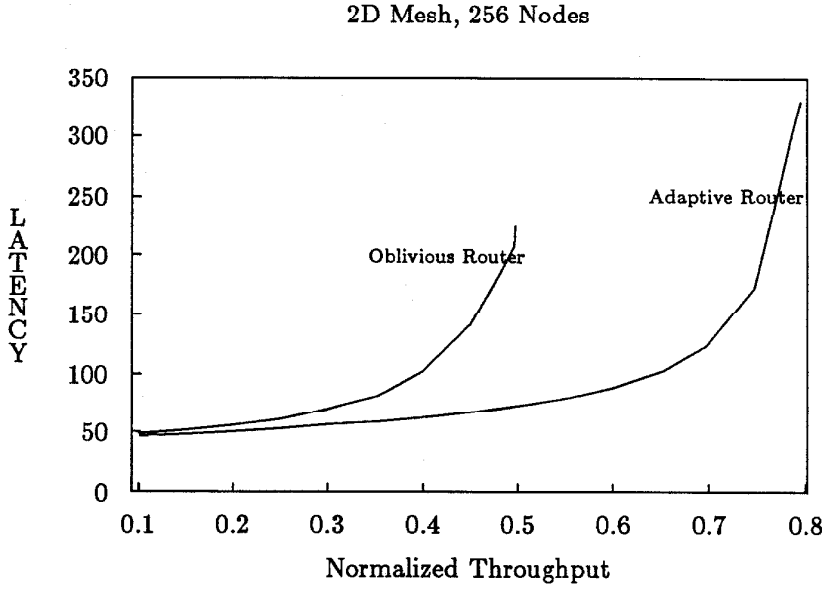

Figure 3.9: Single-Packet Message Latency of 2D Mesh

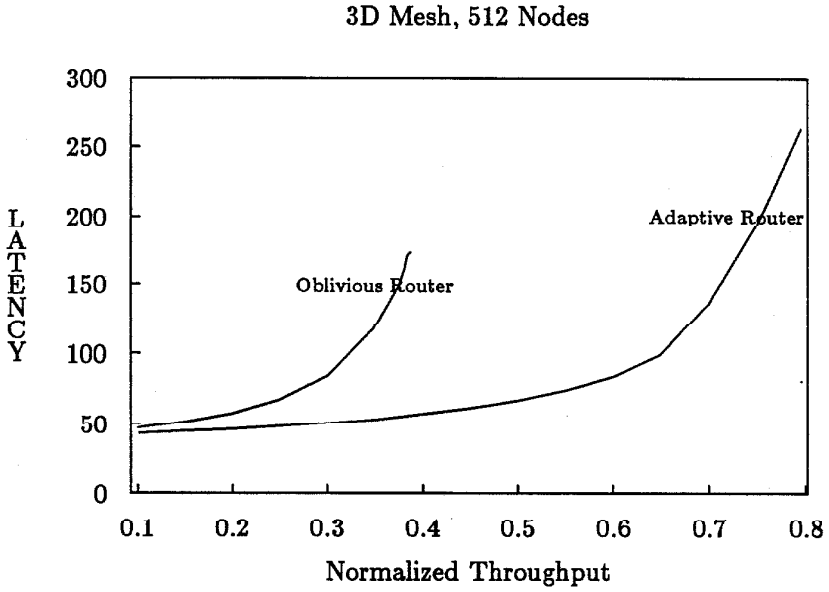

Figure 3.10: Single-Packet Message Latency of 3D Mesh

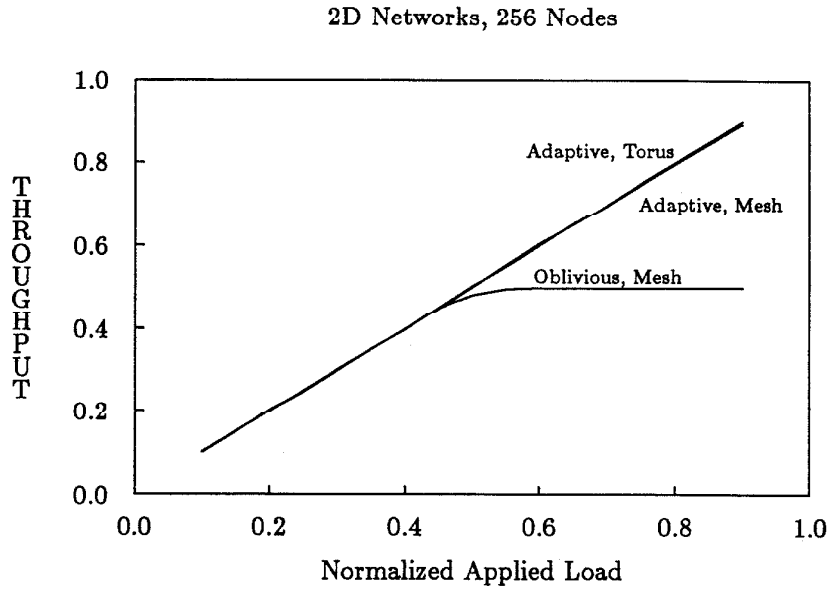

Figure 3.11: Single-Packet Message Throughput for 2D Networks

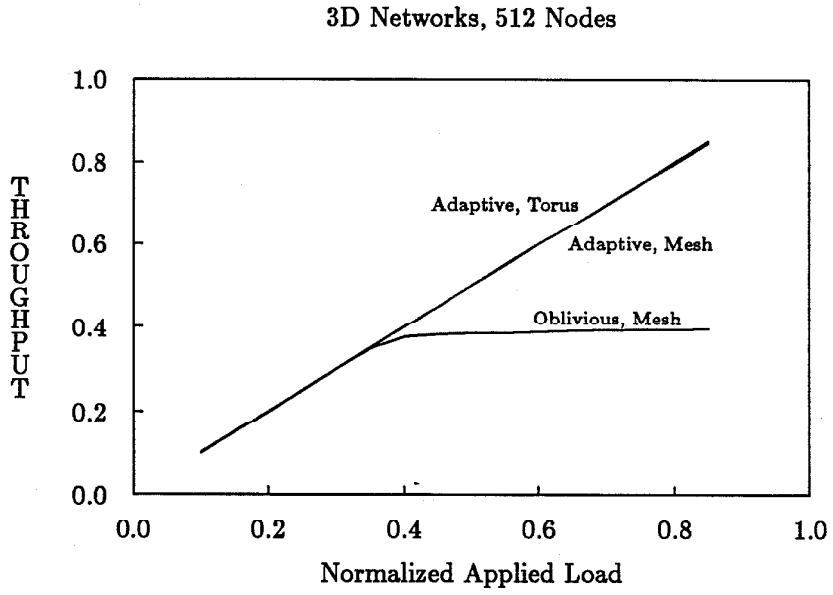

Figure 3.12: Single-Packet Message Throughput for 3D Networks

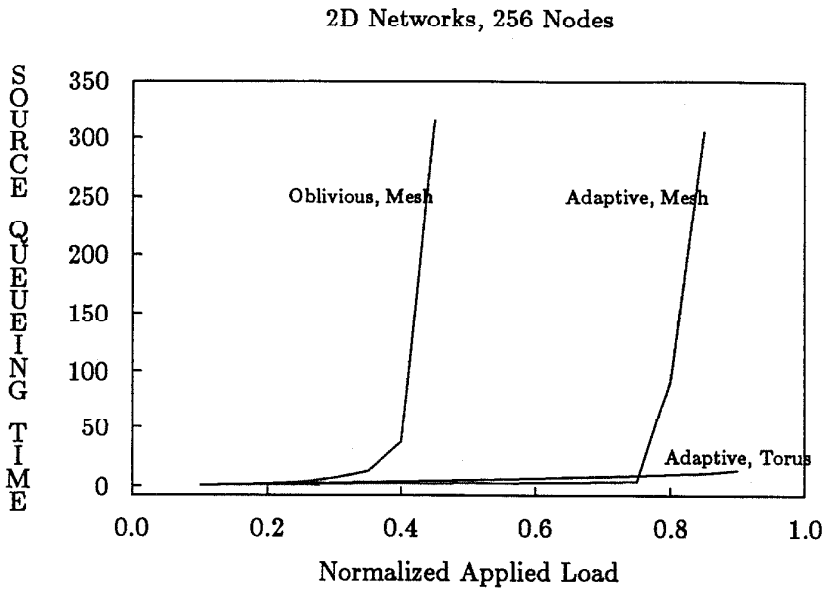

Figure 3.13: Single-Packet Message Source-Queueing Time for 2D Networks

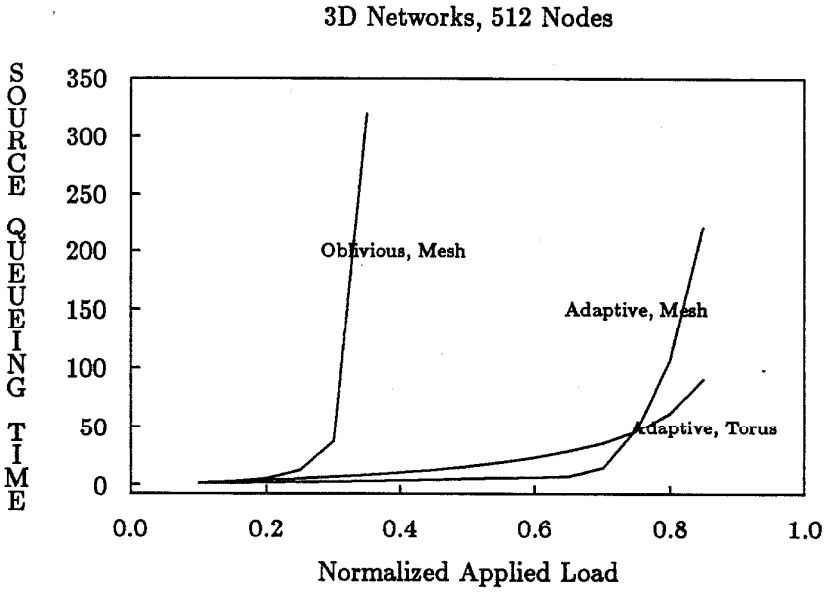

Figure 3.14: Single-Packet Message Source-Queueing Time for 3D Networks

3.5.2 Variable-Length Multipacket Messages

The relevant statistics for the simulation runs of the variable-length multipacket message-traffic experiments are shown in Figures 3.15 to 3.28. The lengths of these messages are chosen according to the *Erlang Distribution* described in the previous section. This generates messages with an average length of 96 flits and a standard deviation of 32 flits. In the oblivious wormhole scheme, the entire message is routed as a single entity across the network; hence, the network sees the same mean and standard deviation of the original distribution. The adaptive cut-through scheme, on the other hand, breaks the messages into a multiple number of fixed-size packets of length 32 flits. As a result of this repackaging process, the network sees a different mean and standard deviation for the length of its message traffic. Results from *Monte Carlo* simulations show that the mean of the repackaged message to be $\approx$ 112 flits, whereas the standard deviation remains $\approx$ 32 flits. This extra overhead reduces the effective maximum throughput to $\frac{96}{112} \approx 85.7\%$ of its original value. In other words, a normalized throughput value of 0.857 represents a *ceiling* on the effective throughput of the adaptive scheme for the message traffic under discussion.

Figures 3.15 and 3.16 plot the latency versus throughput tradeoff curves for the 2D and 3D mesh networks. Shown in these figures are the *message* latency curves for both the oblivious wormhole and adaptive cut-through schemes. The latency curves of individual *packets* in adaptive routing are also plotted. We again observed the familiar characteristic shape of latency versus throughput tradeoff. The latency curves for the oblivious wormhole scheme closely resemble those of the corresponding single-packet message traffic, with the transition points at $\approx$ 30 and 35%, and the maximum sustained throughput at $\approx$ 40 and 50% normalized values, respectively, for the 2D and 3D-mesh. Similarly, the transition points for the adaptive schemes occur at $\approx$ 60 to 65% normalized throughput. The maximum sustained effective throughput is, as far as the figures show, up to at least 80%, or $\approx$ 93% in *actual* network traffic, given a total of 15 packet buffers per node. The difference between the packet- and message-latency values for the adaptive scheme remains relatively fixed over the range below the transition point. In fact, close examination of the numerical figures reveals that it is very close to the difference between the average message length, 112, and the packet length, 32. Furthermore, these packet latency curves are much steeper than the corresponding curves obtained in the single-packet message-traffic experiments. ***In other words, while the adaptive scheme allows exploitation of multiple alternate routes to destination, this advantage appears to be realized only at the message level, but not by individual packets.*** In fact, the very bursty packet traffic generated by these multipacket messages is highly correlated. In a crude sense, successive packets of the same message follow each other sequentially in order, and are broken only occasionally by other competing messages along the way.

Figures 3.17 and 3.18 plot the corresponding latency curves for the 2D and 3D torus networks. The curves for the 2D torus behave as expected, with the transition points lying at $\approx$ 70% in normalized value. Similarly, the maximum sustained effective throughput is at least 80%, the equivalent of $\approx$ 93% in actual network throughput values, given 15 packet buffers per node. The latency curves for the 3D torus, however, are unexpected. While we still recognize the familiar characteristic shape, it is distorted toward the high end. In particular, the maximum sustained effective throughput occurs

at $\approx$ 60% in normalized value. This is much lower than the corresponding maximum obtained in the single-packet experiments, and, *contrary* to intuition, since we generally would expect a 3D network to perform better than its 2D counterpart. This discrepancy is illustrated most clearly in Figures 3.19 and 3.20, which show the curves of throughput versus applied load for all the relevant networks and routing schemes. The maximum sustained throughput under heavy applied load remains very stable for the oblivious wormhole schemes. The throughput curves for the 2D torus, 2D mesh, and 3D mesh under the adaptive scheme all behave normally and do not level off until they are very close to the throughput ceiling. The curve for the 3D torus, on the other hand, starts to level off unexpectedly at $\approx$ 60% normalized value. To understand the reason behind this anomaly, we recall that there is an internal channel in each node that connects the router and the message interface and that it forms a separate *arrival queue* in tandem with the router of the receiving node. For all other network topologies and for their respective sustained traffic density, this queue remains short and its effect can generally be neglected. However, for the 3D torus, since the maximum steady-state injection rate, and, hence, the delivery rate, is identical to 1, this queue will grow to become much longer as the sustained throughput increases. Figures 3.21 and 3.22 plot the average packet population per node versus the applied load for the adaptive routers. The curve for the multipacket 3D torus clearly reveals the rapid growth in queue size. As the normalized applied load approaches 0.6, all available buffers become filled and remain practically full thereafter. Since misrouting of packets is used in the adaptive scheme to prevent deadlock in the network as the buffers become full, the effective normalized throughput does not increase beyond 0.6. The interesting point to observe is that *the effective throughput also does not decrease below 0.6,* as the corresponding throughput curve in Figure 3.20 remains relatively flat under increasingly heavy applied load. Apparently, the misrouting of packets through the internal channel that occurs at those traffic densities is also sufficient to throttle any further congestion created by excessive injection. Figure 3.22 also reveals a similar but less prominent growth in queue size for the case of single-packet messages in a 3D torus.

Figures 3.21 and 3.22 also reveal another interesting point: The queue sizes for the multipacket message traffic are always larger than the corresponding ones for the single-packet message traffic. For example, the 2D mesh at 0.7 normalized applied load, *ie*, around the transition points on the latency curves, has an average queue size of $\approx$ 0.5 for the single-packet message traffic, $\approx$ 3.2 for the multipacket message traffic. Similarly, the average queue size at 0.7 normalized applied load for the 3D mesh is $\approx$ 1.2 for the single-packet message traffic and $\approx$ 4.2 for the multipacket message traffic. We can understand this phenomenon qualitatively in the following way: We recall from the previous discussion that the very bursty nature of multipacket traffic and the correlation in the destination addresses result in channel-access collision patterns that actually more closely resemble *message switching* than packet switching. Hence, theoretically, the average number of queued *messages* per node will remain close to that found in single-packet message traffic; and the number of queued *packets* per node will be much higher, as each message consists of a multiple number of packets. Most importantly, this explains why the maximum sustained throughput is only 0.6 for multipacket messages on the 3D torus. Since, in that case, the queues for normal network traffic and for arrival messages are both competing for the finite number of buffer at each router. In

contrast, the curves from the other network topologies indicate that the extra demand on buffer space due to the arrival message queue is minimal for practical networks of reasonable size. For example, the average queueing population per node for all the other networks is ≈ 6 at 75% applied load and throughput. In any case, the results of these experiments suggest that *the average length of the intended message traffic should be included as an important design factor in determining the exact amount of buffer storage allocated in each node.* On the other hand, as the traffic experiment on the 3D torus shows, *the effect of having insufficient buffers per node appears simply to truncate and create a plateau rather than a disastrous drop in the throughput performance curve.*

Figures 3.23 and 3.24 plot the average source queueing time versus the applied load for all relevant networks and routing schemes. As expected, the curves indicate that the source queueing time remains negligible until the applied load approaches the maximum throughput limit identified in the previous curves, after which it climbs rapidly. The exact numerical values near the high end limits should *not* be taken seriously, as most of them actually increase without bound when one waits long enough. For example, in the case of the 3D mesh, the plotted curve shows a finite average even for applied loads exceeding the throughput ceiling (which is clearly the result of impatience on the part of the author).

Finally, average number of packets and messages being *reassembled*, and *resequenced*, against the normalized throughput for all four different networks. Reassembly of received packets is necessary to deliver entire messages as complete entities; resequencing of received messages is necessary to maintain the message order between senders and receivers. Both operations put extra demand on the local memory of each processing node. However, as is indicated by these curves, the amount of extra storage occupied by these operations appears to be minimal. For example, even at 80% normalized throughput (except the 3D torus, which never reaches 80%), the average number of waiting packets is < 7. In fact, the average fraction of messages received out of sequence is universally $< 0.6\%$, even at 80% sustained throughput. We have chosen not to separately plot this statistic because it is almost entirely flat and stays very close to zero. We conjecture that the main reason behind this *flatness* is that in all our experiments, the profitable channels are always chosen from those that lie along the shortest paths joining the node under consideration and the packets' destinations. As a result, no matter what the local decisions are, every message between the same source/destination pair has to travel an identical number of hops before arriving at its destination. This appears to be sufficient to constrain the set of *probable* message arrival sequences to those very close to the original injection sequence.

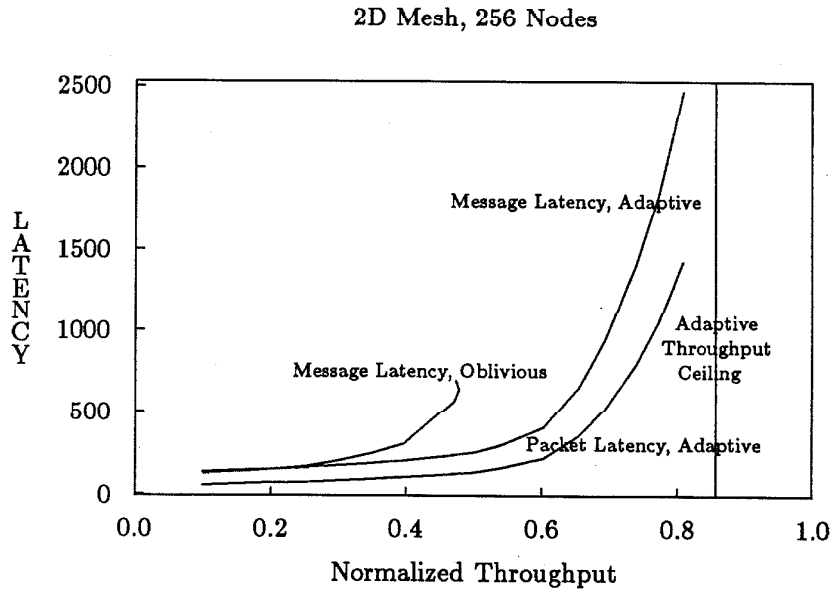

Figure 3.15: Variable-Length Message Latency for 2D Mesh

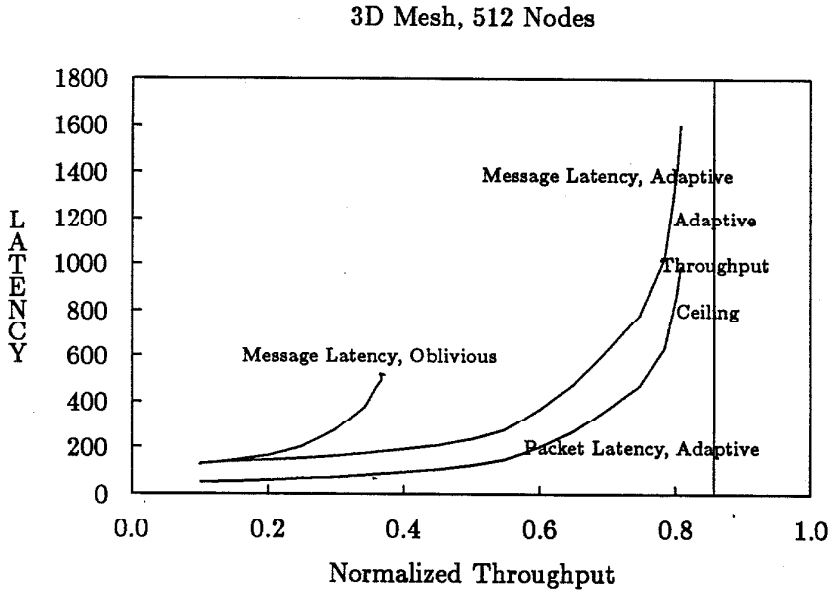

Figure 3.16: Variable-Length Message Latency for 3D Mesh

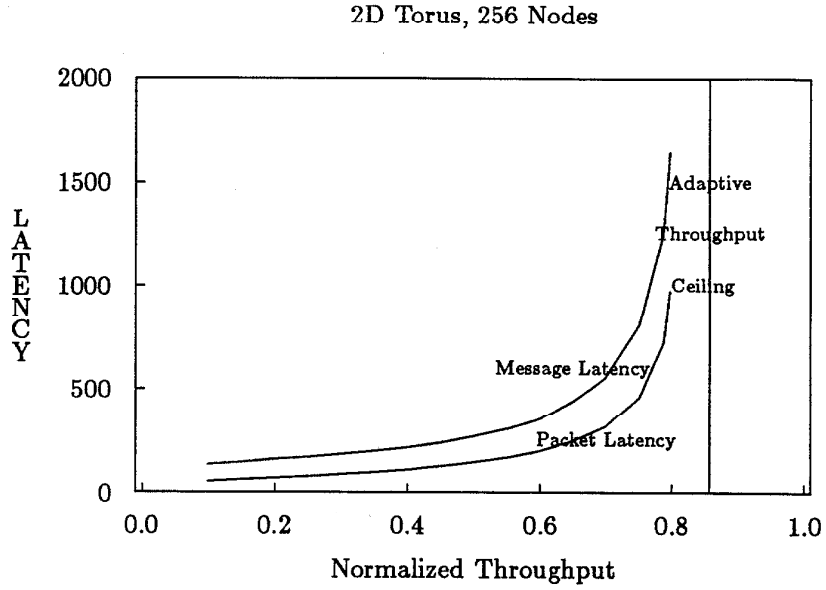

Figure 3.17: Variable-Length Message Latency for 2D Torus

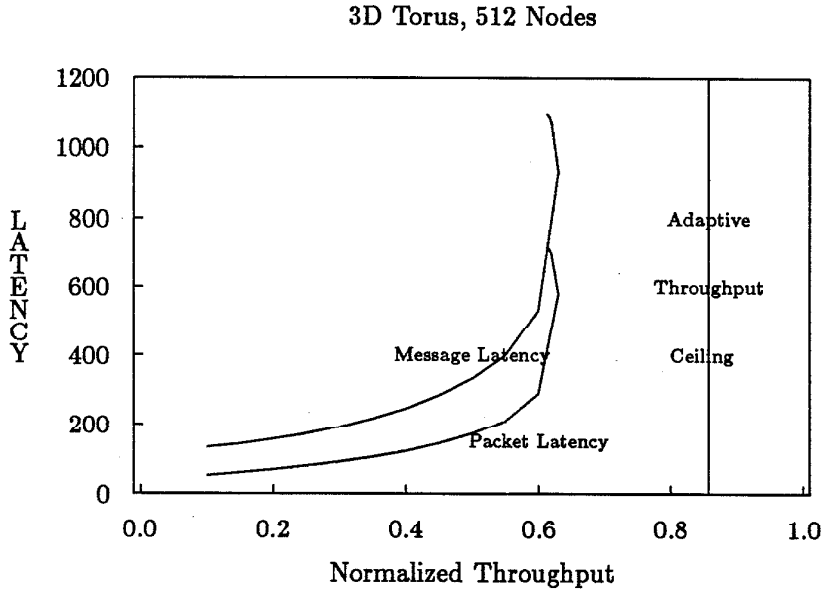

Figure 3.18: Variable-Length Message Latency for 3D Torus

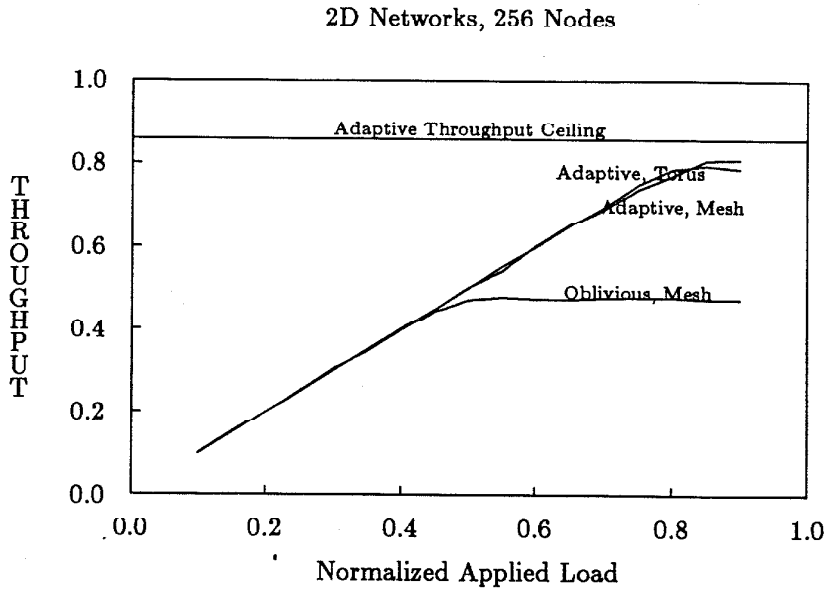

Figure 3.19: Variable-Length Message Throughput for 2D Networks

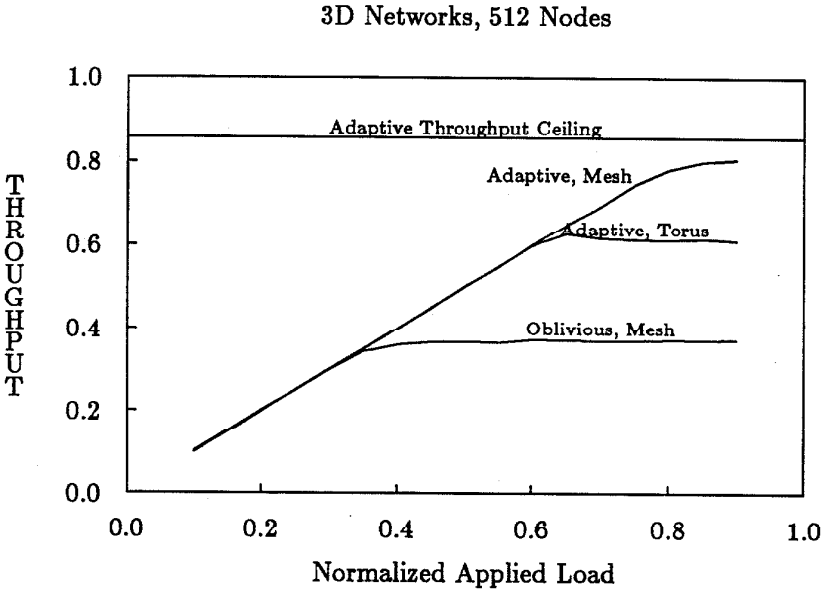

Figure 3.20: Variable-Length Message Throughput for 3D Networks

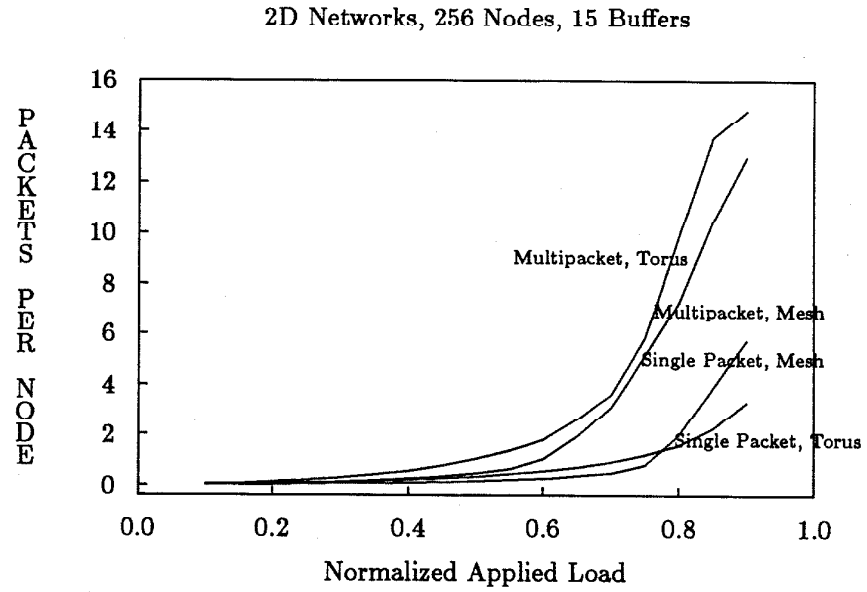

Figure 3.21: Average Adaptive-Router Queue Population for 2D Networks

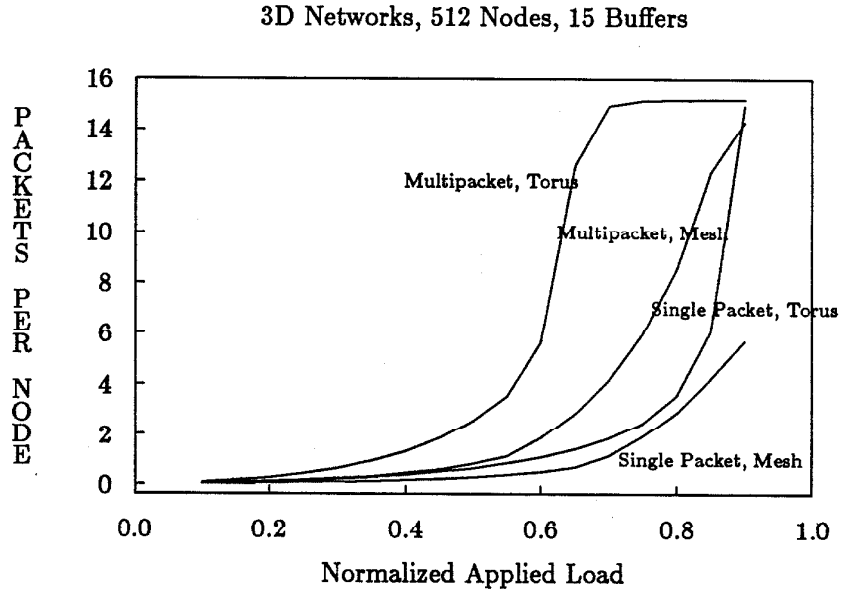

Figure 3.22: Average Adaptive-Router Queue Population for 3D Networks

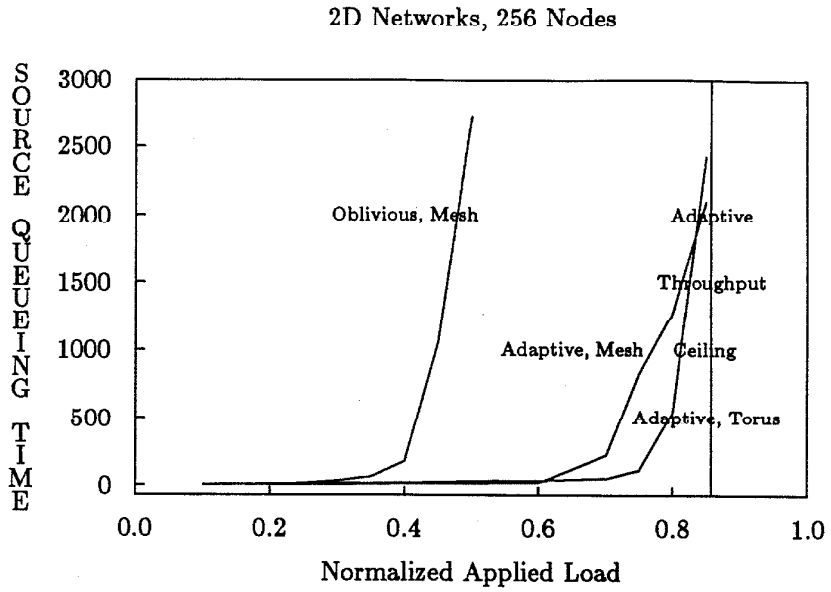

Figure 3.23: Variable-Length Message Source-Queueing Time for 2D Networks

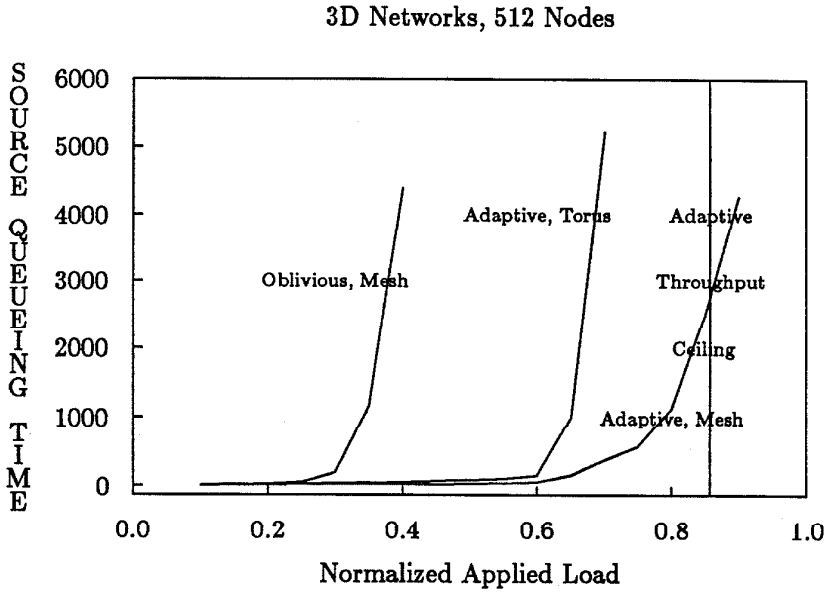

Figure 3.24: Variable-Length Message Source-Queueing Time for 3D Networks

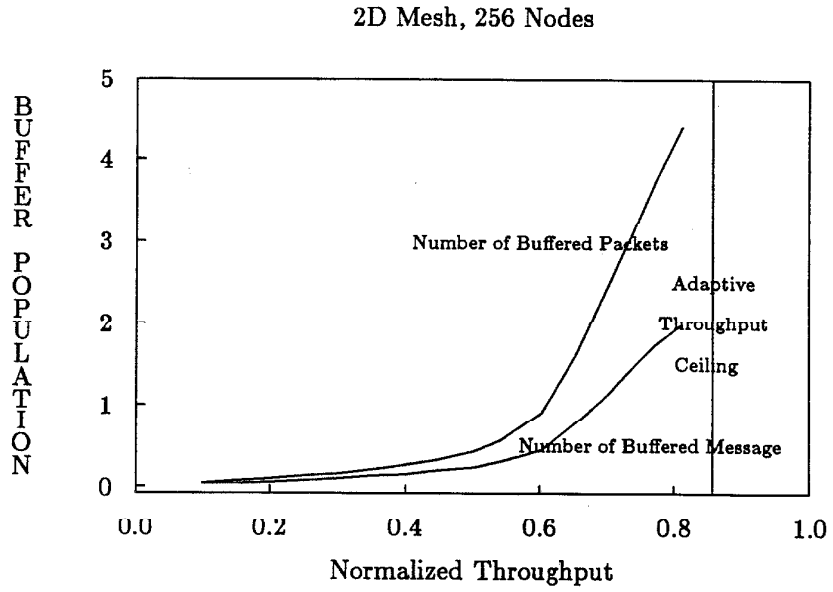

Figure 3.25: Average Reassembling/Resequencing Buffer Population for 2D Mesh

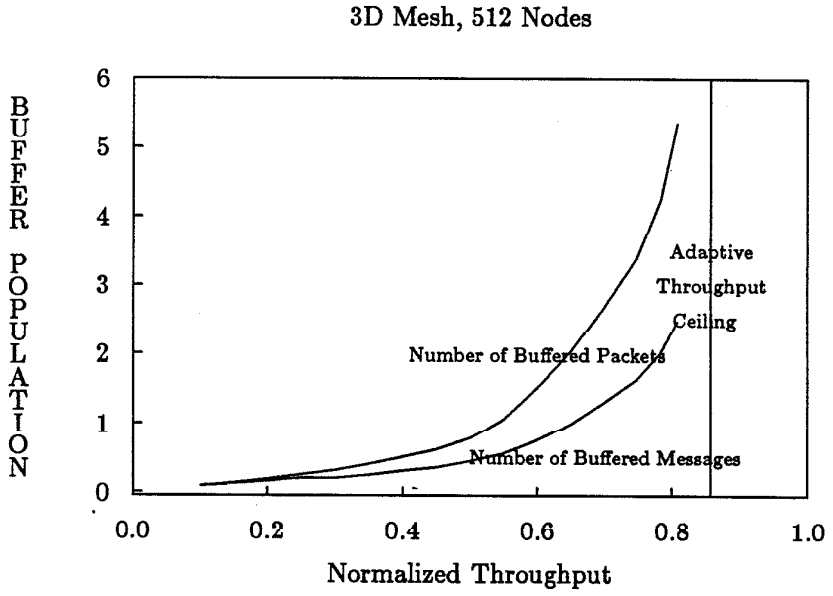

Figure 3.26: Average Reassembling/Resequencing Buffer Population for 3D Mesh

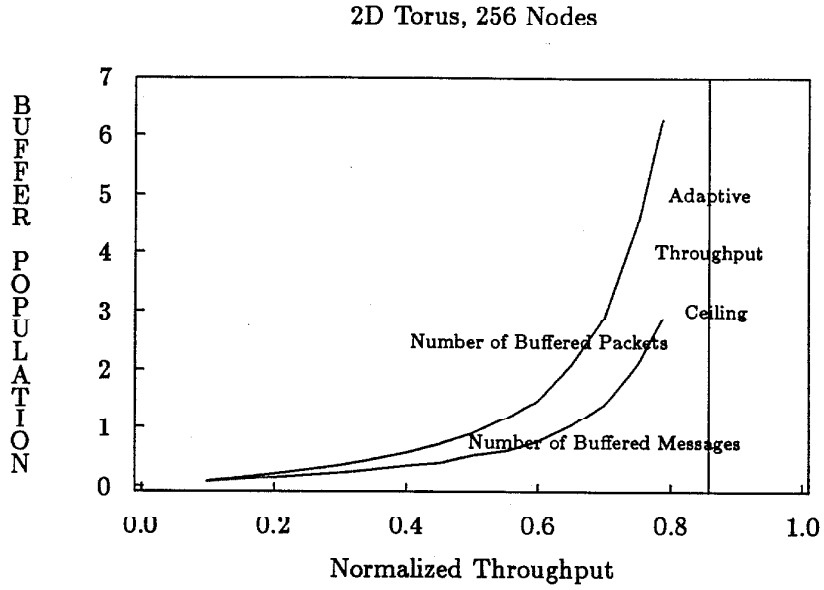

Figure 3.27: Average Reassembling/Resequencing Buffer Population for 2D Torus

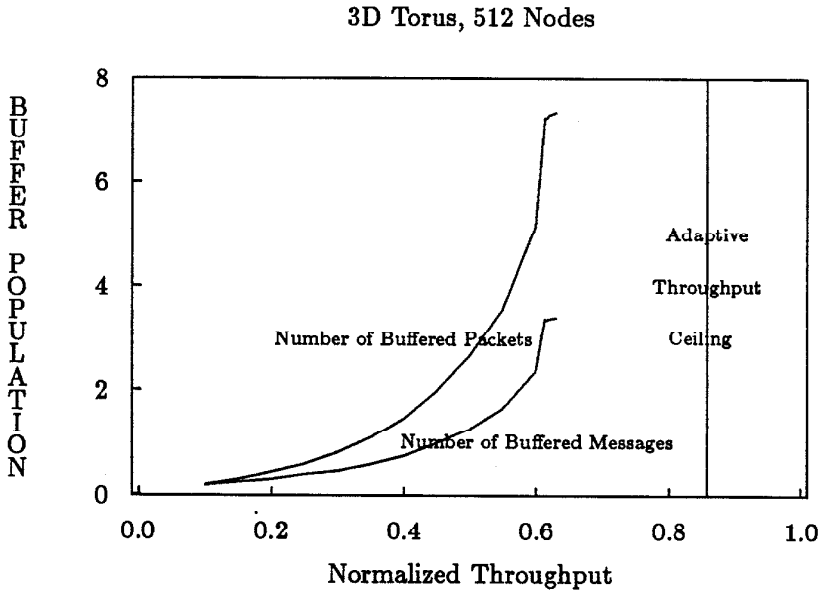

Figure 3.28: Average Reassembling/Resequencing Buffer Population for 3D Torus

3.5.3 Reactive Message Traffic

The relevant statistics for the simulation runs of the reactive message traffic experiments are shown in Figures 3.30 to 3.41. The length of the messages used in this set of experiments are chosen from the same *Erlang Distribution* used in the variable-length multipacket message experiments. In other words, a normalized value of 0.857 again represents the effective throughput ceiling for the adaptive-routing schemes. In these experiments, each node is initially endowed with a fixed, identical number of messages waiting to be processed while the network is quiescent. The amount of time spent in processing each message, referred to below as the *processing delay*[2], is directly proportional to the length of the message. The proportionality constant is a parameter that varied in these experiments. Upon completion of the processing of an old message, a new message is generated with its destination to some other random node. In this way, the total message population in the entire system (network + nodes) is held constant throughout the simulations. This reactive traffic is believed to be a realistic approximation that models the communication patterns and captures the feedback stabilization inherent in most message-passing concurrent computations.

Given an average message length of 96 flits, the average processing delay per message chosen in our simulations spans a range from $\approx$ 100 to 1200 cycles. This range roughly corresponds from one processing cycle per flit to twelve processing cycles per flit, or an equivalent processing rate of from $\approx$ 0.08 flit per cycle to one flit per cycle. Although highly dependent on the programming style, this range was chosen to reflect typical figures obtained in fine-grain concurrent computations. In particular, a number of recent developments have the potential to push the average processing delay toward the extreme low end. The notion of *message-driven processor* [3,13] represents architectural attempts to reduce the substantial overhead in message handling. Furthermore, the very fine object granularity achievable in concurrent languages employing the *actor* computation model [1,3,7] encourages a programming style that generates computations with extremely simple objects and very intensive message traffic; preliminary traffic statistics obtained from programming experiments using Cantor [3,7] produce results with extremely low processing delay. In addition, the disparity between the typical channel flit width and the typical processor word width also contribute to reduce the average processing cycles per flit.

Figures 3.30 and 3.31 plot the average sustained *network throughput* versus the average *processing rate* for the 2D mesh under the oblivious wormhole and adaptive cut-through schemes, respectively. Corresponding curves for the 3D mesh, the 2D torus, and the 3D torus are shown in Figures 3.32 to 3.35. Shown in each of these figures are four curves, corresponding to the different starting message populations of 1, 2, 4, and 8 messages per node. In each of these plots, a higher starting message population always results in a higher sustained throughput. In fact, a population of eight messages per node is sufficient to drive each network to the maximum throughput capability achievable under the respective employed routing scheme. For each curve shown, starting from a very low processing rate, the average throughput increases roughly linearly as the average processing rate increases, until it reaches saturation, and stays constant thereafter. Under adaptive cut-through routing, the transition occurs at an average

[2]Not to be confused with the packet *routing* processing delay mentioned in section 3.1.

processing rate of ≈ 0.2 flits per cycle for the 2D mesh and ≈ 0.4 flits per cycle for the 3D mesh. These rates are approximately equal to the maximum sustained throughput obtained from our previous simulation results. To understand these figures, observe that as one increases the average processing rate, the rate at which a processor can generate messages increases. *The transition points occur roughly at the point where a processor working full speed can produce a traffic density high enough to saturate the network throughput capacity under the employed routing scheme.* Below the transition rate, however, a processor simply cannot produce messages fast enough to cause saturation. As a check, we observe that the transitions occur at processing rates of ≈ 0.12 flits per cycle for the 2D mesh and ≈ 0.2 flits per cycle for the 3D mesh, using oblivious wormhole routing. Because of its very high throughput capacity, the curves for the 3D torus exhibit no transition for the range of delay values shown here.

Figures 3.36 to 3.41 plot the *processor utilization ratio* versus the average message *processing delay* for the various networks and routing schemes. The various utilization curves all start from very low ratio and climb roughly linearly until they reach saturation. Again, the saturation utilization ratio for the $m = 8$ curves are always higher than the rest. In fact, with $m = 8$, the saturation values are almost always unity. Observe that the transitions into saturation also occur at roughly the corresponding throughput saturation points. For delay values less than the saturation figure, the utilization ratios are much lower than one. The very fast turnaround time from message consumption to message generation at these lower processing delay values produces little or no processing backlog at each node. On the other hand, the bottleneck occurs in front of the communication network, which is unable to deliver messages fast enough to keep the processors busy. As the processing delay values increase beyond the saturation value, a backlog of workload starts to pile up at each node. As a result, the processor is always working at full speed trying to catch up with existing backlog. The communication network, operating at less than full capacity, readily replenishes each node with new backlog arriving from other nodes. These observations are consistent with the average message source-queueing time and processing backlog-time statistics collected during the experiment. In general, as the processing delay increases, the source queueing time starts from very high and decreases to almost negligible; in contrast, the processing backlog time starts from almost negligible and increases to very high values. In summary, we can conclude that, according to our experiments, with a sufficient number of computing objects per node, a computation with an average message-processing time that is less than the saturation value is *communication time bounded.* On the other hand, a computation with an average message-processing time that is greater than the saturation value is *processing time bounded.* This suggests that *the ideal message processing rate is approximately the rate at which the messages generated are just enough to saturate the network.* The particular saturation values will depend on the network parameters and routing schemes employed; and, in practice, they will also depend on the message-destination distributions. As the network size increases and resource per node decreases, maintenance of message locality becomes progressively more important. Exactly how to achieve load balance while retaining message locality at a *reasonable* cost remains a difficult issue.

Another interesting point to observe is that as the number of starting messages per node is reduced, the corresponding sustained network throughput and processor

utilization are also reduced. For example, except at very low delay values under oblivious wormhole routing, none of the $m = 1$ curves ever reaches throughput saturation; and none ever fully utilizes the processors. In fact, at high delay values, the utilization ratio for the $m = 1$ experiments stays relatively flat at ≈ 0.55, irrespective of the network topology or routing scheme used. Similar *asymptotic* behaviors on the processor utilization ratios are observed for all other message populations as well, regardless of the underlying topology and routing scheme. In retrospect, we see that it is possible to understand these asymptotic equilibrium behaviors. In fact, we shall derive a very simple, closed-form analytic approximation of the asymptotic processor-utilization ratio for any finite message population, based on certain simplifying assumptions:

- First, we note that at the very high processing delay values where the asymptotic behavior is observed, the network throughput is extremely low, and the average message latency is *negligible* when compared to the average processing delay. Hence, we can obtain a very good approximation by regarding message delivery as *immediate*. This also explains why the asymptotic behaviors are identical across different network topologies and routing schemes.

- Next, we make the simplifying assumption that the message processing delays are *exponentially* distributed, although it was actually *Erlangian* in our experiments.

- The third assumption we make is to assume that the number of nodes in the network, N, is very *large*. Having a very large network allows us to equate the *time average* obtained at a node to the *ensemble average* obtained over the network.

- Another assumption we make is to assume that the total message population in our network is very large. Hence, it is possible, if improbable, for a node to have a very large backlog. This allows us to obtain very good approximation by replacing the finite *total* population restriction with the finite *average* population restriction.

- Finally, we observe that since the random message destinations are generated uniformly over the entire network, this gives rise to complete homogeneity and allows us to focus the analysis on a single node.

Let S_i and p_i denote the state, and its corresponding probability, of having a backlog of i messages waiting at each node, including the one being processed. The desired processor-utilization ratio is then simply given by $1 - p_0$. Let μ denote the average message-processing rate; this is also equal to the message-generation rate at each node having a nonempty backlog. Given the network in equilibrium, the average fraction of nodes with nonempty backlog is given by $1 - p_0$. Also, since each message generated has an equal probability of being destined to any other node, the average message arrival rate at each node is given by $(1 - p_0)\mu N \frac{1}{N} = (1 - p_0)\mu$. These considerations allow us to write down the following set of equilibrium equations (see Figure 3.29):

$$\begin{aligned} p_0(1 - p_0)\mu &= \mu p_1 \\ p_1(1 - p_0)\mu &= \mu p_2 \\ &\vdots \\ p_i(1 - p_0)\mu &= \mu p_{i+1} \end{aligned}$$

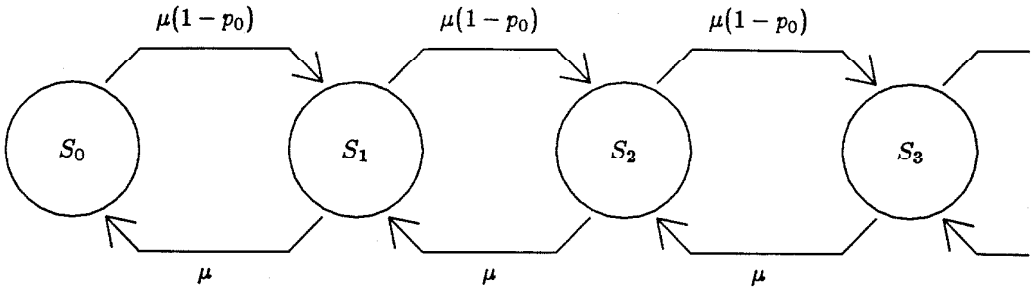

Figure 3.29: State Transition Rate Diagram for Asymptotic Utilization Analysis

subjected to the normalizing constraint of $\sum_{i=0}^{\infty} p_i = 1$. The regularity of the above set of equations allows us to solve for p_k explicitly in terms of p_0:

$$p_k = p_0(1-p_0)^k$$

Observe that the normalizing constraint is trivially satisfied independent of the value of p_0. To determine the exact value of p_0, we need to use the fact that the network consists of a closed system of $M = m \times N$ messages. Given the immediate message-delivery assumption, as well as the homogeneity condition, each node in equilibrium with the rest of the network will have an expected number of backlog that is exactly equal to m. In other words, we have:

$$\sum_{i=0}^{\infty} i p_i = m$$

Using the obtained explicit form for p_k, we obtain:

$$\begin{aligned} \textstyle\sum_{i=0}^{\infty} i p_i &= p_0(1-p_0)\textstyle\sum_{i=1}^{\infty} i(1-p_0)^{i-1} \\ &= p_0(1-p_0)\tfrac{1}{p_0^2} \\ &= \tfrac{1-p_0}{p_0} \\ \Longrightarrow \qquad p_0 &= \tfrac{1}{m+1} \end{aligned}$$

This gives us the desired closed-form analytic approximation for the asymptotic processor utilization ratio:

$$U_{\text{asymptotic}} = \frac{m}{m+1}$$

For the values $m = 1, 2, 4$ and 8, the approximation gives utilization ratios of 0.50, 0.67, 0.80, and 0.89. These figures are all within $\approx 5\%$ of the experimentally obtained figures, and, more importantly, this derivation reveals the nature of this asymptotic behavior. On the one hand, the analytic approximation and experimental results show that for *irregular* computations whose communications lack any discernible pattern one cannot realistically expect 100% efficiency, even in the extreme case when communication delay is negligible. On the other hand, it is consistent with the intuitive belief that decomposing a computation into more objects of finer grain size will increase the amount of concurrency. This increase in concurrency is subsequently reflected in a higher average processor utilization ratio.

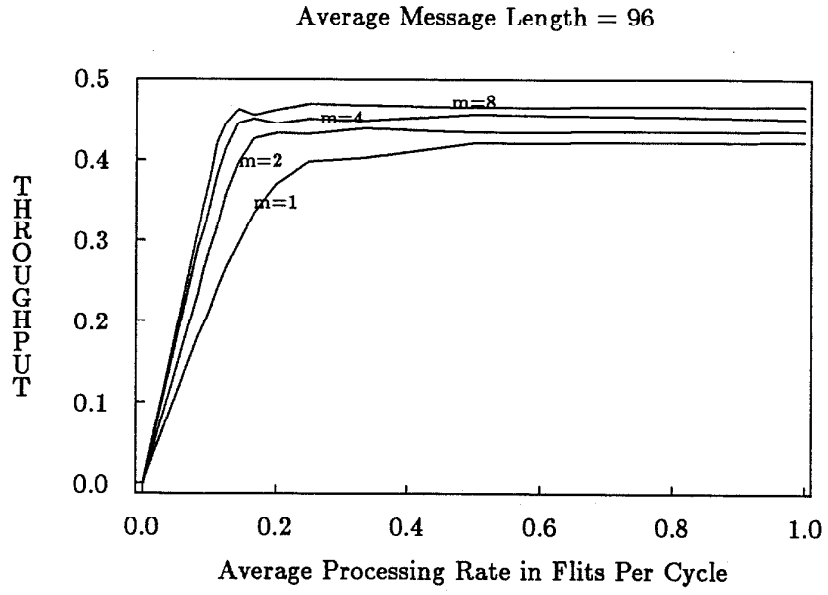

Figure 3.30: Oblivious Network Throughput under Reactive Traffic for 2D Mesh

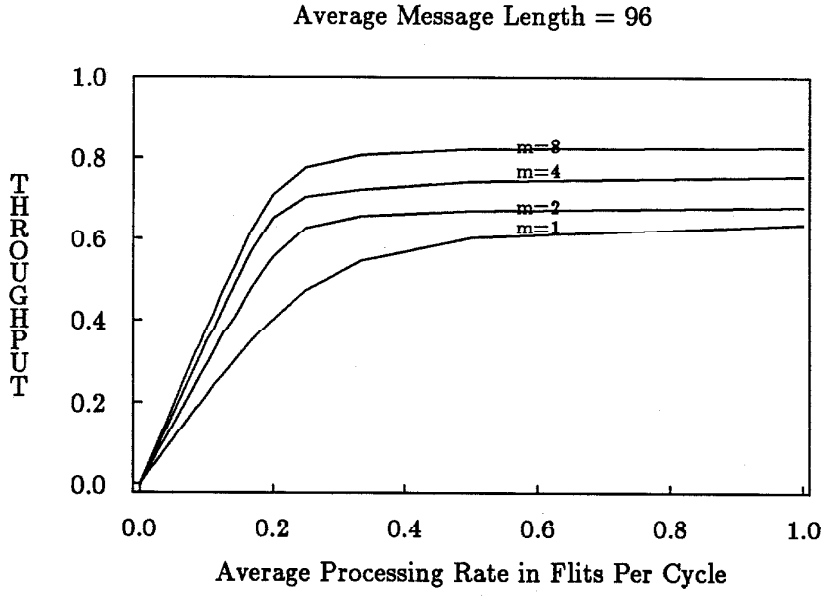

Figure 3.31: Adaptive Network Throughput under Reactive Traffic for 2D Mesh

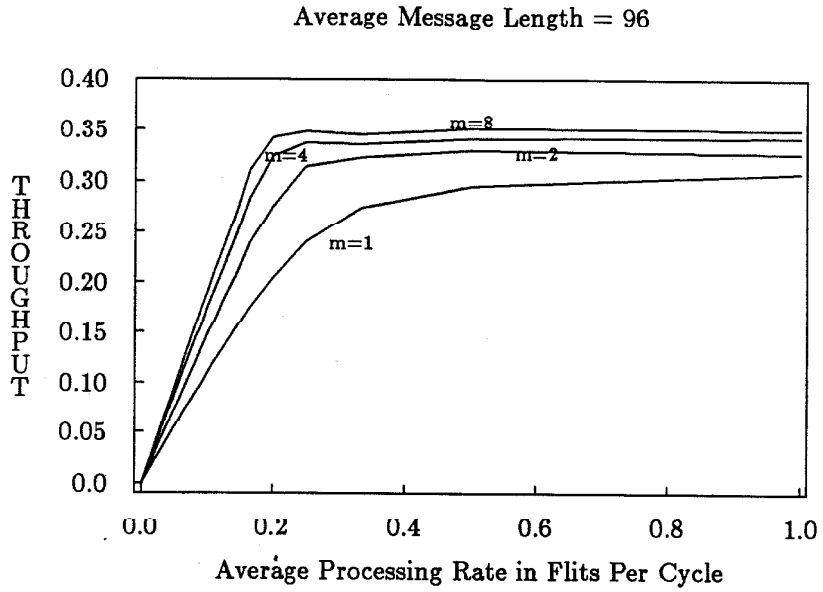

Figure 3.32: Oblivious Network Throughput under Reactive Traffic for 3D Mesh

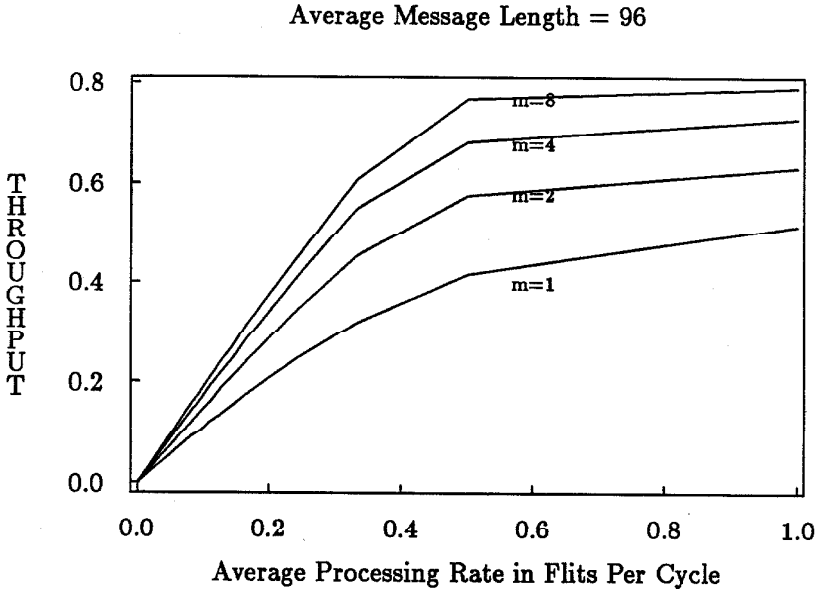

Figure 3.33: Adaptive Network Throughput under Reactive Traffic for 3D Mesh

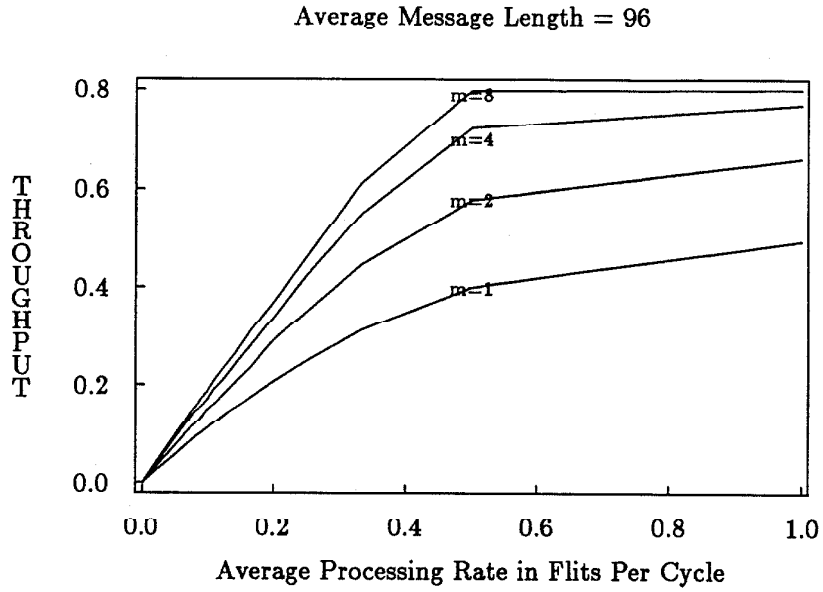

Figure 3.34: Adaptive Network Throughput under Reactive Traffic for 2D Torus

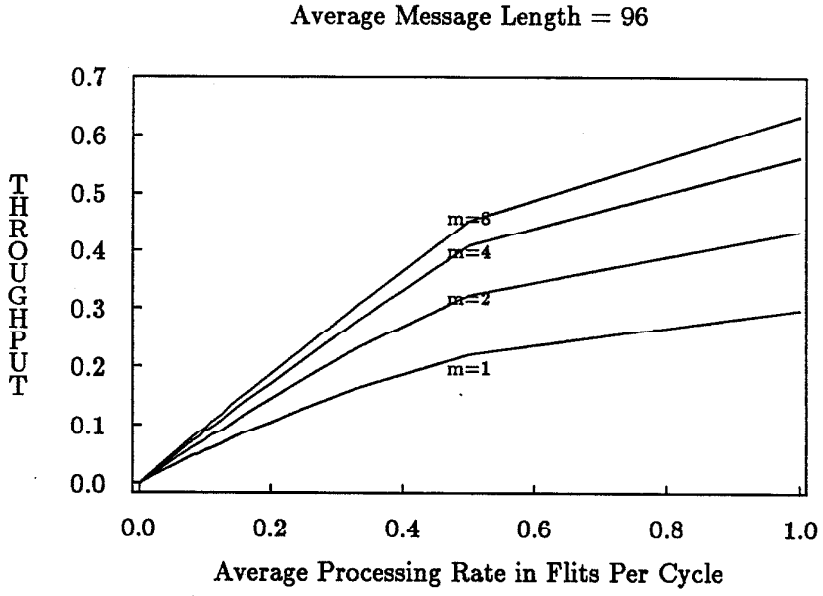

Figure 3.35: Adaptive Network Throughput under Reactive Traffic for 2D Torus

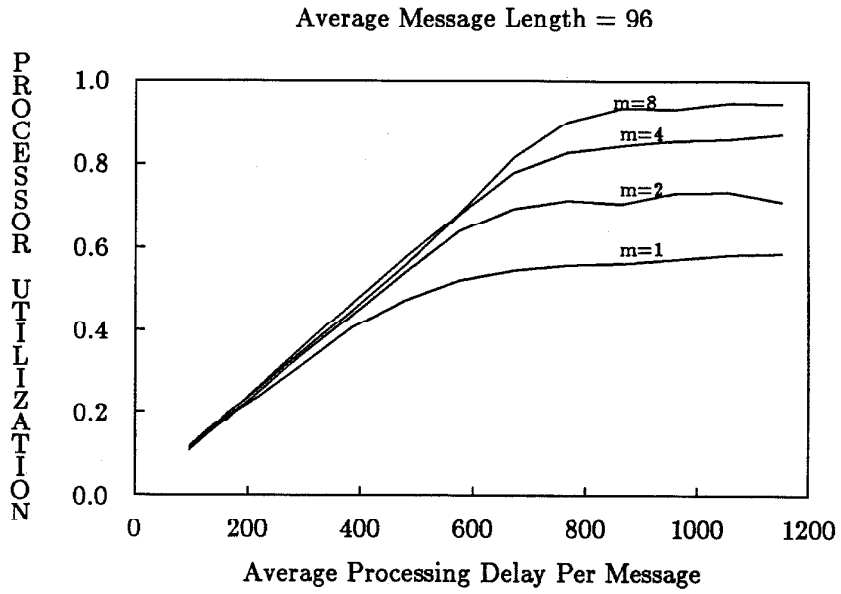

Figure 3.36: Oblivious Processor Utilization under Reactive Traffic for 2D Mesh

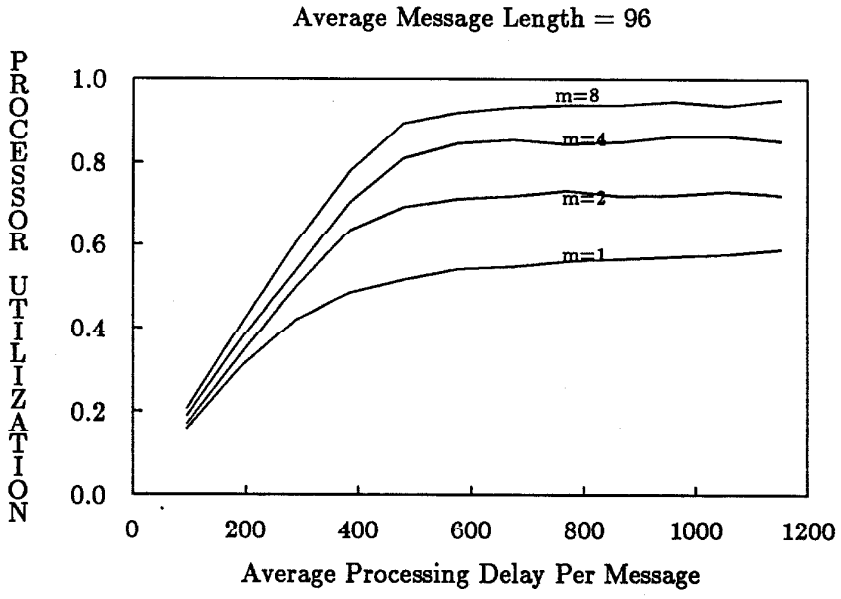

Figure 3.37: Adaptive Processor Utilization under Reactive Traffic for 2D Mesh

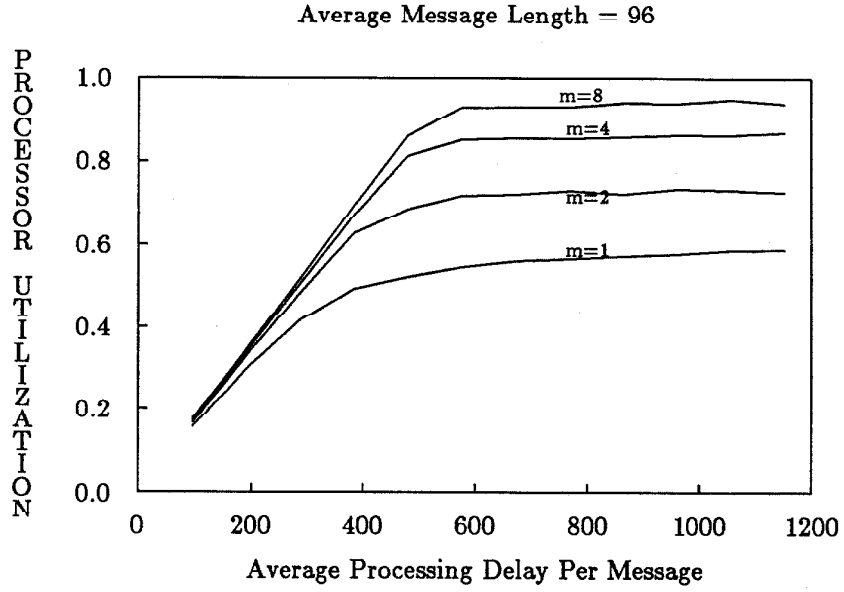

Figure 3.38: Oblivious Processor Utilization under Reactive Traffic for 3D Mesh

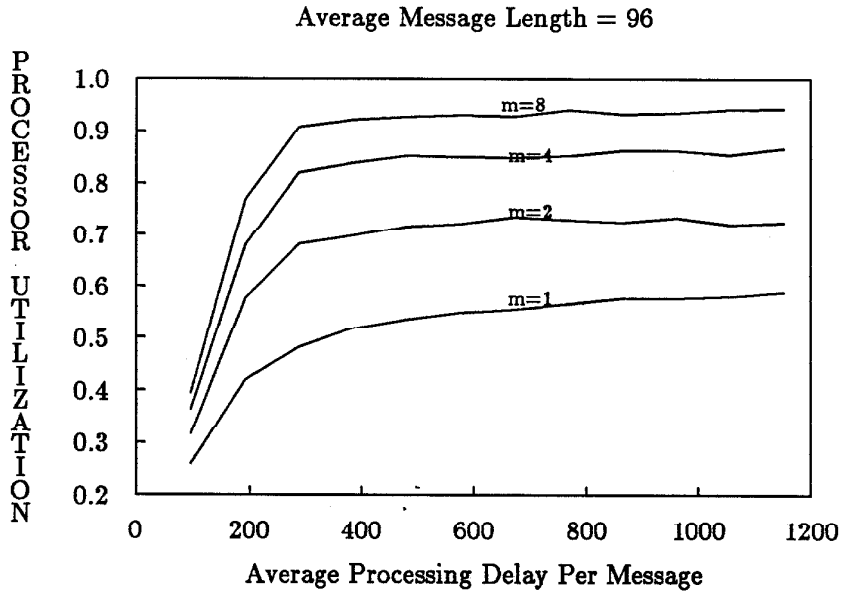

Figure 3.39: Adaptive Processor Utilization under Reactive Traffic for 3D Mesh

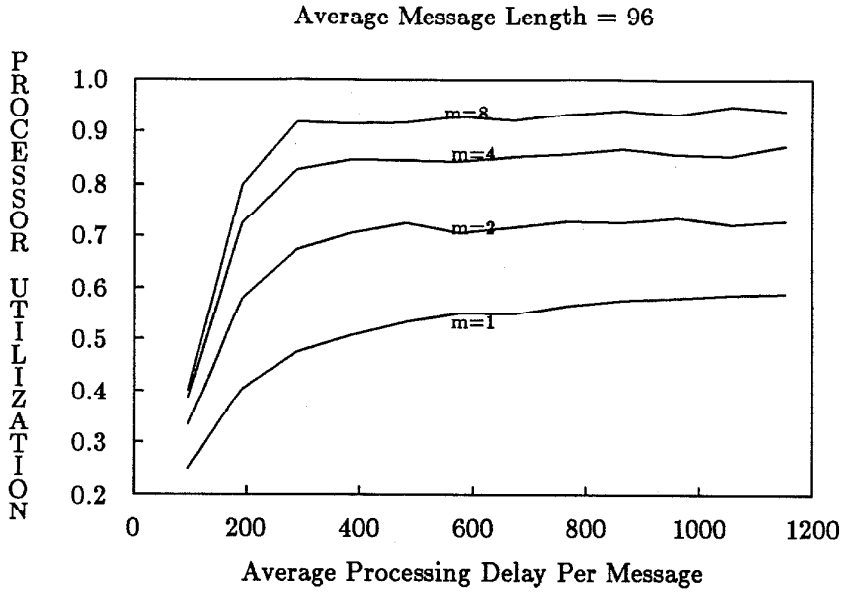

Figure 3.40: Adaptive Processor Utilization under Reactive Traffic for 2D Torus

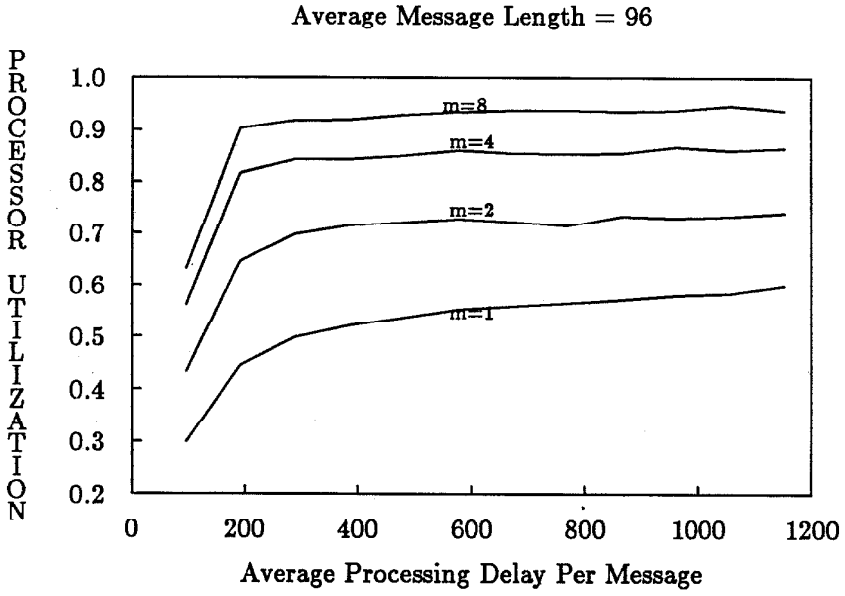

Figure 3.41: Adaptive Processor Utilization under Reactive Traffic for 3D Torus

3.5.4 Congestion-Controlled Message Traffic

The relevant statistics for the simulation runs of the congestion-controlled message traffic experiments are shown in Figures 3.42 to 3.53. In these experiments, additional congestion control, also known as *flow control* [17], in the form of injection restriction has been imposed. The restrictions are coordinated through the use of the injection-guarantee protocol described in section 2.5.3. In its original form, the fairness protocol assures eventual network access whenever a node has a packet queued for injection; in a certain sense, this already provides limited congestion control. The protocol used in these experiments is an extension that provides more spontaneous congestion control: A node that is forced to misroute its buffered packets will not advance its injection count until after all currently misrouted packets have completely left, irrespective of whether it has a packet to inject or not. The rationale for this added restriction is that misrouting of any sort will serve as a good indication for the onset of congestion.

The performance curves from all the previous experiments show that as the network is pushed toward its throughput limits, the quality of the first-order performance metric, *eg*, message latency, decay rapidly. In addition, there are the *second-order* metrics, which are at least of equal importance. These include the standard deviation of the message latency, where a large value implies that there are large fluctuations from the average. This is undesirable, and, hence, a main objective for imposing congestion control is to confine the network operating points to the region where it will deliver acceptable performance. In fact, since all performance metrics display a transition of qualities, with the transition points occurring roughly around the same throughput value, the congestion-control mechanism should ideally prevent the network from passing beyond the transition point, yet not interfere at lower throughput values. This set of experiments tests the effectiveness of our congestion-control protocol against this objective.

Figures 3.42 to 3.45 plot the average sustained network throughput against the average applied load under adaptive cut-through. Two curves are shown in each of these figures: One is from our earlier uncontrolled simulations, where channel-access conflicts were resolved according to the distance priority. For these, our earlier experiments show that the latency transition points all occur at $\approx$ 70 to 75% normalized throughput. The second one is the congestion-controlled curve where conflicts are resolved in a first-come first-served fashion. It is remarkable to observe that, in every one of these figures, except for the single-packet message traffic in the 2D mesh, the controlled throughput curve levels off almost exactly at the respective transition value. Furthermore, for an applied load below the transition, there is no difference between the controlled and the uncontrolled curves: Both climb along with unit slope. Similarly, for applied load beyond the transition, the throughput remains stable without showing any reduction. The controlled saturation throughput for multipacket messages in a 3D mesh is at $\approx$ 67%; and is consistent with our expectations following the earlier discussion in section 3.5.2 because of the relatively small radix (8) of the chosen 3D network. On the other hand, the corresponding controlled saturation throughput for single-packet message traffic in the 2D mesh is $\approx$ 90% normalized value. This is $\approx$ 10% higher than expected. To understand this will require an examination of the corresponding latency curves.

Figures 3.46 and 3.53 plot the average message latencies versus the normalized throughput for single-packet messages. In both cases the controlled curves are consis-

tently lower than the corresponding uncontrolled curves around the transition region; *eg*, in the 2D mesh, at 80% throughput, the average latency of the controlled traffic is $\approx \frac{1}{2}$ that of the uncontrolled traffic. Similar but less prominent latency improvements are observed in the 3D mesh as well. Such latency reductions are not surprising, but are in fact consistent with the nature of our congestion-control mechanism. To see this, observe that misrouting has the side effect of ***undoing*** the useful work done previously. Therefore, to keep up with the demanded throughput, the channel utilization and, hence, the network population will both have to increase beyond that found without misrouting. The net result of this increased network population is an increase in latency, as predicted by Little's theorem [30]. Restraining new injections whenever misrouting occurs under our congestion-control protocol keeps the amount of misrouting minimal; thus keeping the network population lower than that found with uncontrolled traffic. This is confirmed by the simulation statistics. This translates into a lower average latency; as a result, the effective transition points are shifted a bit higher, explaining the higher saturation value observed in Figure 3.42 for the 2D mesh under congestion control. Similar latency reductions around the transition regions are also observed in Figures 3.50 and 3.52 for the multipacket message traffic.

The corresponding standard deviations of these message latencies are shown in Figures 3.47, 3.49, 3.51 and 3.53. Again, substantial reductions are observed in all the controlled traffic around the respective transition regions. Note that the observed standard deviations are always much lower than the corresponding average latencies for the controlled traffic, whereas they are approximately identical for the uncontrolled traffic. A reduction in the latency standard deviations implies a more consistent routing quality for all the packets delivered by the network, and this is achieved without using priority assignments. This improved consistency can be understood as follows: While the distance priority employed in the uncontrolled traffic assures global progress, such assurance does not apply to individual packets. In fact, packets that are far away from their destinations are more likely to be misrouted as traffic density increases. This in turn further increases the latencies of these *long-distance* packets, which translates into much higher latency standard deviations. In contrast, for the controlled traffic, channel-access conflicts are resolved in the first-come first-served manner. In the absence of misrouting, the FCFS scheme assures *fairness* in channel access, given a dynamically changing collection of packets. By waiting long enough, each packet is guaranteed assignment to one of its profitable channels, regardless of its distance from destination. While such an assurance no longer holds when packets are misrouted, the nature of the congestion control protocol is such that it will help minimize the possibility of misrouting. As a result, this assurance remains *approximately* valid, thus leading to a reduction of the latency standard deviations.

In summary, the experimental data obtained in these simulations suggest strongly that the described congestion-control scheme is very effective in achieving the desired objective, which is to confine the network operating points within regions that will deliver acceptable network performance. In particular, it is very effective for preventing the network from passing the transition point, while not interfering at lower throughput values. As we shall see in Chapter 5, this congestion-control scheme offers a practical alternative to the rather complicated priority-based scheme for providing network progress assurance.

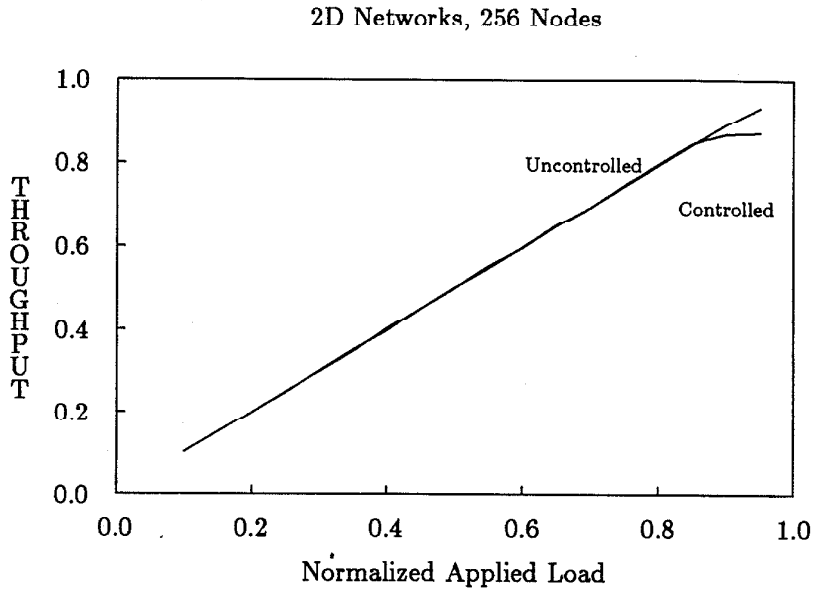

Figure 3.42: Throughput Comparison for Single-Packet Messages in 2D Mesh

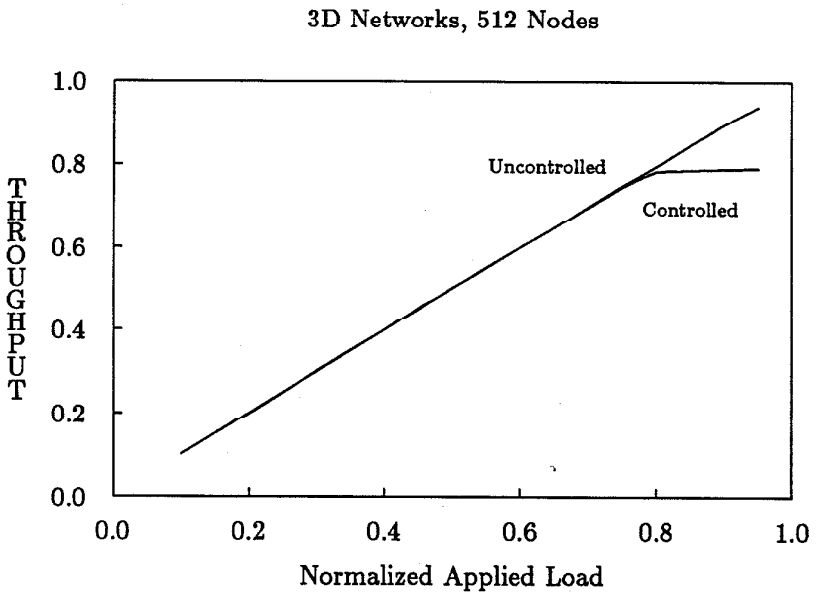

Figure 3.43: Throughput Comparison for Single-Packet Messages in 3D Mesh

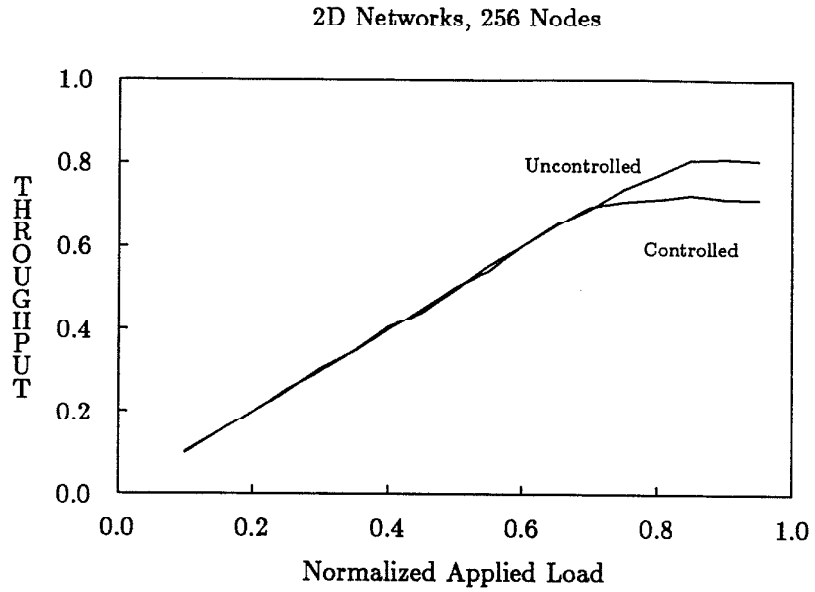

Figure 3.44: Throughput Comparison for Multipacket Messages in 2D Mesh

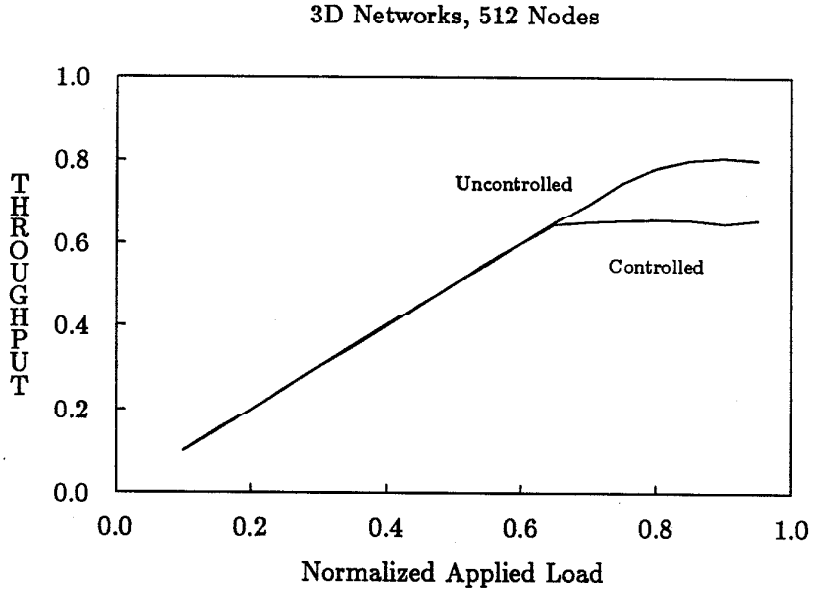

Figure 3.45: Throughput Comparison for Multipacket Messages in 3D Mesh

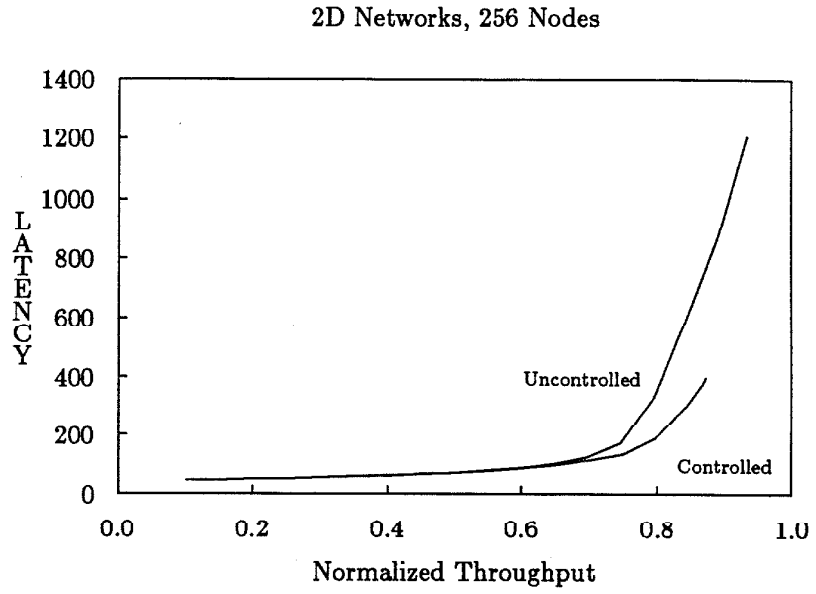

Figure 3.46: Latency Comparison for Single-Packet Messages in 2D Mesh

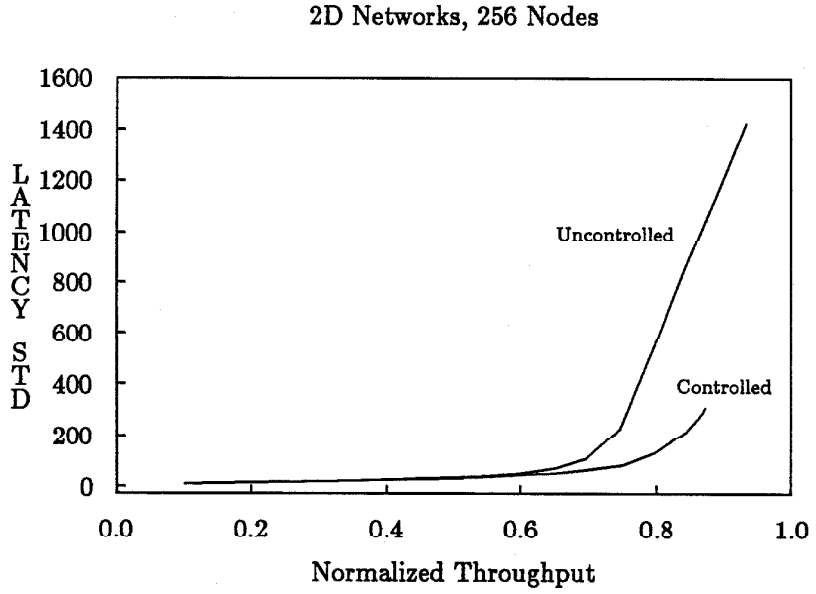

Figure 3.47: Comparison of the Standard Deviations of the Latencies for Single-Packet Messages in 2D Mesh

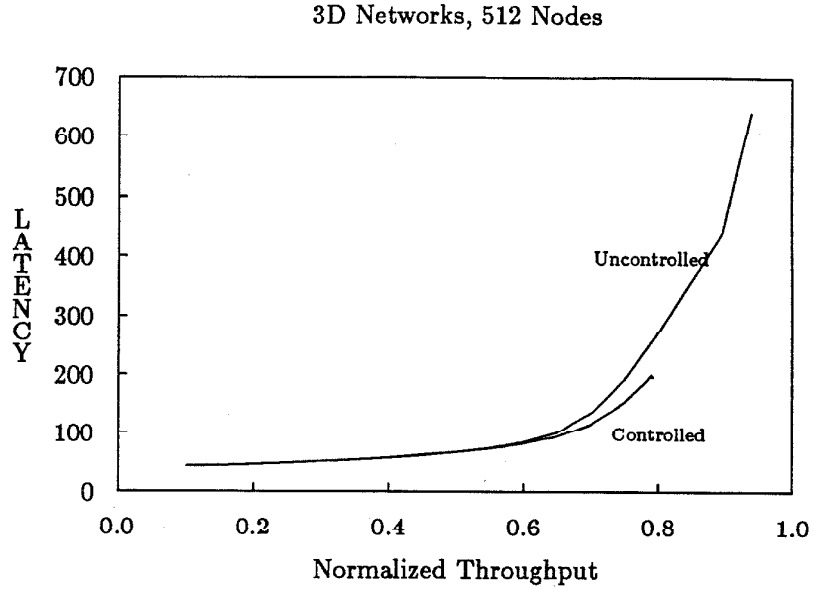

Figure 3.48: Latency Comparison for Single-Packet Messages in 3D Mesh

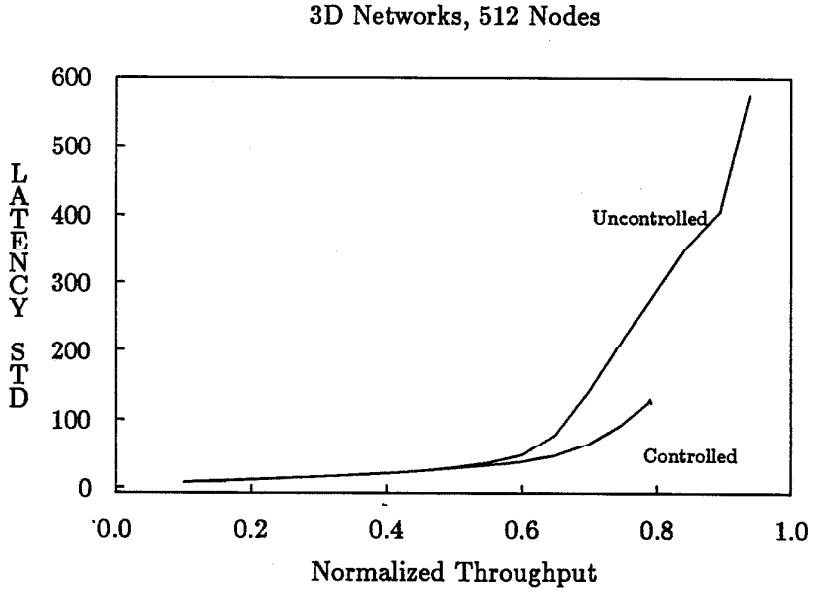

Figure 3.49: Comparison of the Standard Deviations of the Latencies for Single-Packet Messages in 3D Mesh

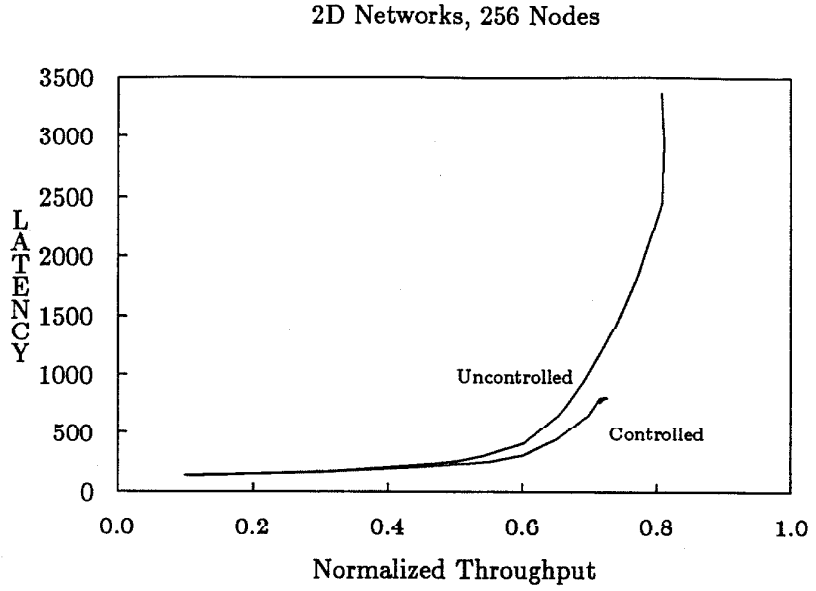

Figure 3.50: Latency Comparison for Multipacket Messages in 2D Mesh

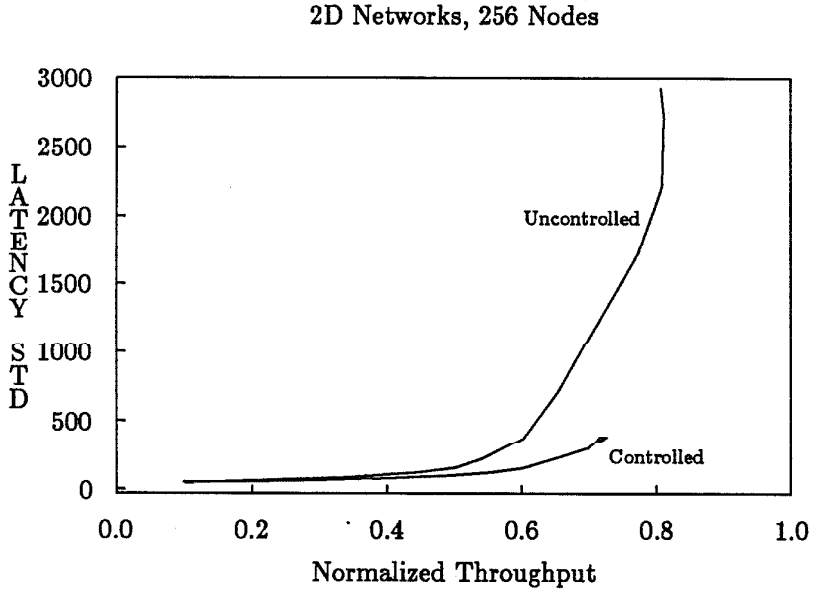

Figure 3.51: Comparison of the Standard Deviations of the Latencies for Multipacket Messages in 2D Mesh

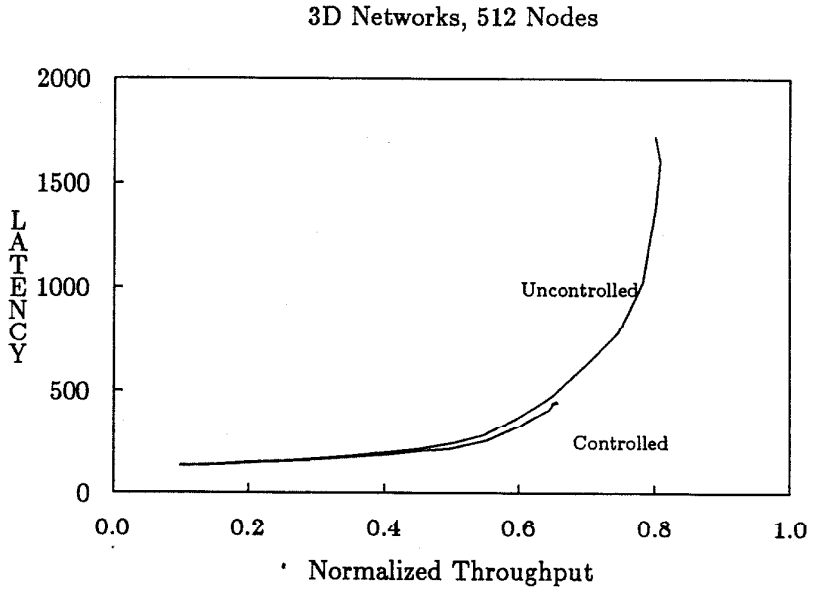

Figure 3.52: Latency Comparison for Multipacket Messages in 3D Mesh

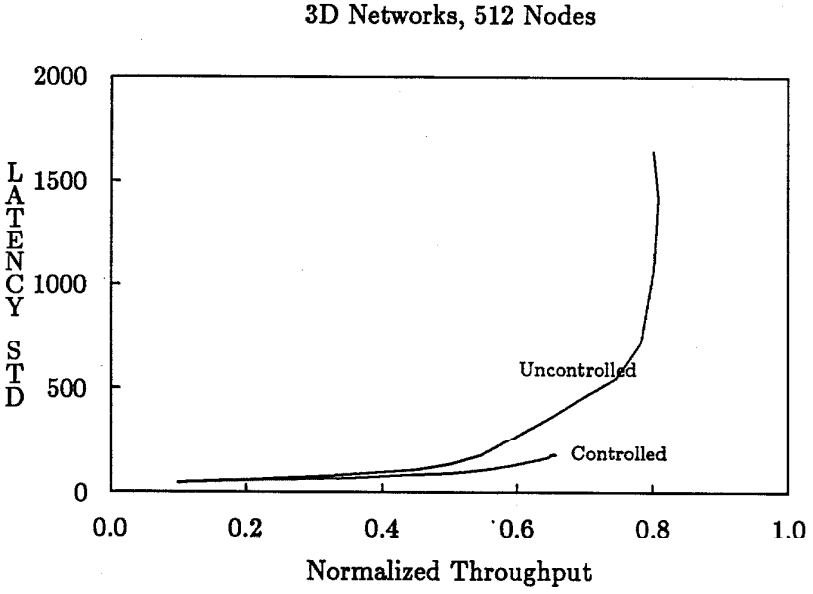

Figure 3.53: Comparison of the Standard Deviations of the Latencies for Multipacket Messages in 3D Mesh

3.5.5 Fast Fourier Transform Traffic

The data dependency graph generated by the FFT computation is shown in Figure 3.54 [50]. The computation is divided into $\log N$ stages, and can be best understood if one associates the data elements with corresponding vertices of a hypercube of the same size. At each stage, intermediate data are generated and distributed between pairs of vertices, known as *butterflies*. The receiving side of the butterflies are comprised of vertices that differ in their *highest* dimension, whereas the sending side are comprised of vertices that differ in *successively lower* dimensions of the imagined hypercube. Since the receiving end of the butterflies always involves nodes that differ in the highest dimension, we have grouped data elements that correspond to vertices that differ in their most significant bits into each node. In this way, every butterfly will both send and receive exactly two messages during each stage. At the end of the computation, each node will hold the transformed values of the *same* locations in the output data array. We have assumed that the various ω^j values have all been precomputed and are stored at each node. The required payload for each message consists of two double-precision floating-point numbers that correspond to the real and imaginary parts of the complex numbers per data point, the physical destination address, and an integer that indicates the message's stage count. At each stage, upon receipt of the two messages, a node has to perform two butterfly operations, each of which consists of four floating-point multiplications and six floating-point additions. This gives a total of twenty floating-point instructions per stage before the generation of two new messages for the next stage. The exact number of cycles required to process these floating-point instructions could differ tremendously, depending on whether the nodes have access to floating-point hardware. For example, on one hand, the newer RISC processors, like the Motorola 88000 family [38], have on-chip floating-point hardware that employs pipelining and can complete one floating-point operation per machine cycle. On the other hand, the processor may have to emulate floating-point instructions completely in software, and typically requires hundreds of instructions per floating-point operation.

For systematic placement, the 12-bit logical index of each data item provides a direct mapping allowing one to physically locate the data item: The least-significant 8-bits of the index gives the physical node identifier to where the data can be found; and the most significant 4-bits differentiate between those stored at the same node. Under this mapping, the average message-to-destination distance is six hops. For the randomized placement, the pseudo-random bijective mapping is generated by a linear feedback shift register sequence that is defined by an 8^{th}-degree primitive irreducible polynomial over GF(2) [35]. In comparison, the particular pseudo-random mapping chosen generates an average message-to-destination distance of 9.6 hops.

The simulation results for the FFT computations are plotted in Figure 3.55. The quantities plotted are the number of butterfly operations per cycle *delivered* by each node averaged over the entire duration of the FFT computation. The experiments were carried out with the network speed held constant while the available computational speed at each node was varied as the independent parameter. In order to relate this abstract quantity to a more familiar notion of speed, let us assume that the network cycle time is 20 *ns*. Then a speed of 0.01 butterfly operations per cycle is equivalent to 5 MFLOPS. Hence, the range of available computing speeds at each node plotted is as high as 12.5 MFLOPS. As a comparison, the Motorola 88000 family of RISC processor

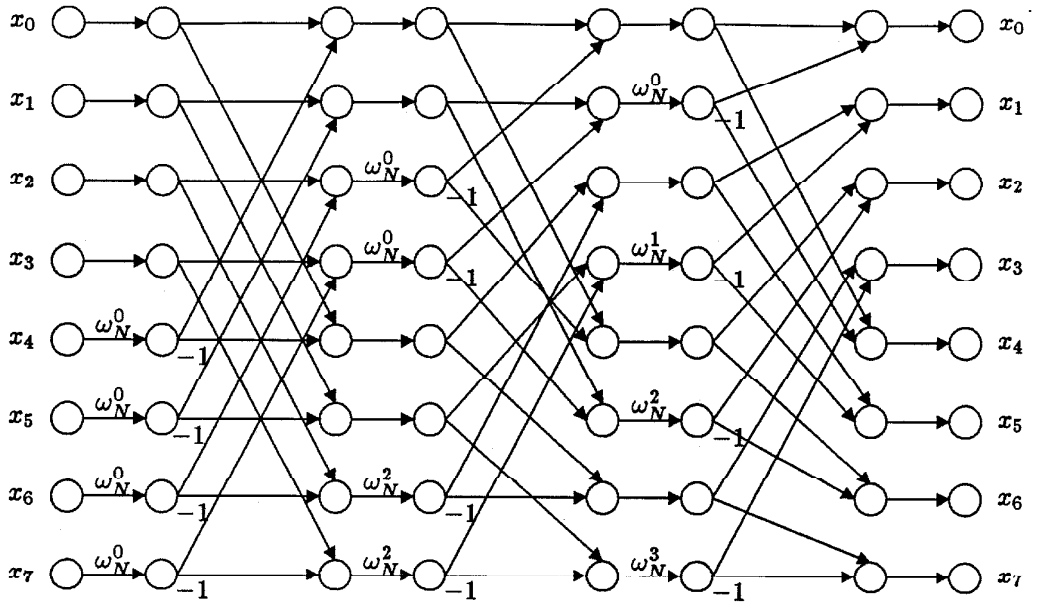

Figure 3.54: The Data Dependency Graph of Fast Fourier Transform.

with an on-chip floating-point unit running on 20 MHz clock claims to deliver $\approx$ 6 MFLOPS performance.

The performance curves for the four experiments all start initially as *unit* slope lines with the average delivered speeds following closely those of the available speed at each node. As the speed of each node keeps increasing, the average delivered speed exhibits a *transition* and then starts to level off. The respective transition points of the oblivious control occur at $\approx$ 2.5 MFLOPS, whereas those of the adaptive control occur at $\approx$ 5 MFLOPS. Roughly speaking, these transition points separate the performance curves into two distinct regions. For a node speed that is *slower* than that of the transition point, the computation is primarily *processing* time bounded. Instead, for node speed faster than the transition point, the computation becomes *communication* time bounded. Observe that as long as one stays within the processing time bounded region, the concurrent formulation enjoys a close-to-ideal *linear* speed-up factor. One advantage of the adaptive scheme over the oblivious one is that it allows the computation to stay inside the linear speed-up region for even faster node speeds.

Another interesting point to observe in these curves is that the performance difference between systematic placement and randomized placement is much more pronounced in the case of oblivious control than in the case of adaptive control. In our experiments, the average distance a message has to travel was 6.0 under the systematic placement and 9.6 under the randomized placement (which represents an increase of over 50%). The two performance curves diverge only after the network has become saturated. This empirical result seems to suggest that the performance of the adaptive scheme is less sensitive than the oblivious scheme to object placement. It should be pointed out, however, that it is premature and risky to generalize on a single example.

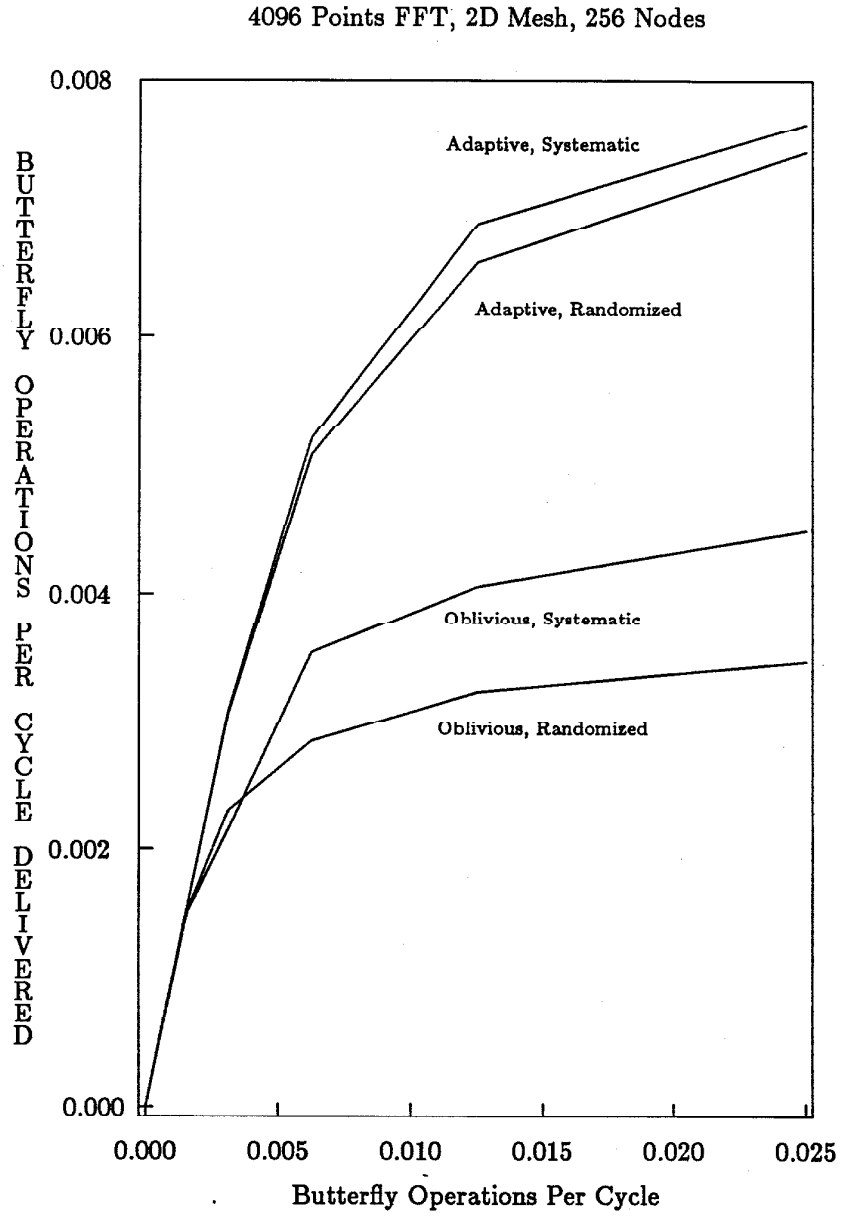

Figure 3.55: Computation Rate per Node — Delivered *versus* Available.

3.6 Summary

In this chapter, we have examined a number of issues concerning the performance of our adaptive cut-through routing formulation in multicomputer networks. The emphasis has been to understand the dynamics governing the different factors and their relationships to the overall performance of the network. Theoretical bounds on the various average performance metrics have been derived for the general class of k-ary-n-cubes and meshes under random message traffic. These bounds provide a uniform frame of reference for the interpretation of performance results obtained under the adaptive routing scheme. In particular, we have studied the performance behaviors both analytically through stochastic modeling, and also experimentally through extensive traffic simulations. We now summarize the major facts and conclusions drawn, based on our studies:

- For random traffic, the average node injection rate scales inversely with the radix of the network.
- Randomized object-placement strategies are simple and very effective in delivering good assignment statistics for the adaptive multipath routing control.
- The message latency obtained under cut-through switching is close to that of circuit switching at low traffic density, and approaches that of store-and-forward switching at higher traffic density. One advantage of the adaptive scheme over the oblivious scheme is the postponement of transition until a higher load is applied.
- The average length of the intended message traffic should be included as an important design factor in determining the amount of buffer storage to be allocated in each router.
- The effect of having insufficient buffers per router is to truncate and create a plateau, rather than induce a disastrous drop, in the throughput curve.
- Just as one would expect, simulations show that for reactive computations, the ideal object-processing rate is approximately the rate at which just enough messages are generated to saturate the network. A faster processing rate results in computations that are communication-time bounded, whereas a slower rate results in computations that are processing-time bounded.
- The congestion-control protocol, which is an extension of the network access fairness assurance protocol in Chapter 2, has proven to be very effective in confining the network operating points within regions that will deliver acceptable network performance. In fact, actual performance improvements have observed around the performance transition regions.
- Although it is premature to conclude, the performance of the adaptive control does appear to be less sensitive than the oblivious scheme to object placements.

In addition to the ones studied in this chapter, there are obviously many other performance characteristics that are also both interesting and informative. Rather than attempting comprehensive coverage, we have chosen instead to focus our investigations on those that we believe are the most important and fundamental. Having studied in

detail the feasibility and performance issues, we shall move on in the next chapter to investigate questions concerning potential reliability gain under the adaptive-routing formulation.

Chapter 4

Reliability

The issues involved in building reliable multicomputers are numerous and complex, and require coordinated efforts across many levels of abstraction, from physical hardware support, through operating system functions and communication protocols, to user programming constructs and methods. The design and building of a reliable communication network is a necessary and important step in this direction. Since it is impossible to build a finite physical network that would always operate correctly in spite of component failures, we have to settle for the more modest alternative of providing fault-tolerant routing in the presence of a *limited* number of failures. Fault-tolerant routing has been an intensively studied subject in the long-haul and local-area network communities, and is gaining more and more attention in the multicomputer/multiprocessor network literature. Some examples of previous work include [4,18,20]; an informative survey can be found in [44].

In this chapter, we investigate and evaluate the potential reliability enhancements achieved by the adaptive routing schemes studied in the last few chapters. In particular, we shall investigate the effectiveness of our adaptive-routing formulation as a *general* technique to exploit the *inherent path redundancies* provided by the richly connected topologies employed in multicomputer networks, such as the k-ary-n-cubes and meshes. We shall focus on addressing issues that are fundamental to performing routing successfully in a faulty communication network. Our primary interest is in the potential of the adaptive approach rather than in any specific networks.

Specifically, in section 4.1 of this chapter, we motivate and discuss the problems involved in performing message routing in a faulty multicomputer network. The discussion identifies and contrasts the special requirements dictated by high performance that are unique in large-scale multicomputers with those faced in conventional networks. Based on this understanding, we then present a set of assumptions and an idealized model that is intended to capture the salient features of faulty multicomputer networks so as to provide the framework for our current discussion. In section 4.2, we present the main conceptual development of this chapter: Two theoretical notions are introduced that characterize the conditions under which our adaptive routing formulation is adequate to provide fault-tolerant routing. In section 4.3, we continue the discussion by describing the computational problems involved in the application of these ideas, and discuss a possible heuristic solution. In section 4.4, we describe a set of simulation experiments, and the corresponding results, that were designed to evaluate the effectiveness of the

adaptive approach for the class of k-ary-n-cube and mesh networks. In section 4.5, we provide an example to illustrate the potential of the adaptive approach: Based on the obtained experimental results, we motivate and suggest the *octagonal mesh* network, a variant of the basic 2D mesh, that shares most of the same advantages of the rectilinear mesh. We also develop a set of routing relations for the octagonal mesh that displays excellent fault-tolerant potential under our adaptive-routing formulation. This section also includes an in-depth study of the various performance and fault-tolerant behaviors of the network. Finally, in section 4.6, we summarize the various developments presented in this chapter.

4.1 Routing in Faulty Networks

In this section, we motivate and discuss the problems involved in performing message routing in a faulty multicomputer network. Before we proceed with the main discussion, we have to be more specific about our scope of interest. In particular, we want to make a distinction between fault-tolerant *routing*, our specific interest, and the much more general study of fault-tolerant *communication*, of which the first can be considered a subtopic. In fact, fault-tolerant communication is a vast subject consisting of many different area of interests. For example, maintenance of *reliable* communication across *unreliable* physical channels has long been an intensively researched area, and many protocols designed to achieve reliable communication have been documented in published literature [56,6]. Similarly, the study of the theory and practice of forward *error-correcting codes* [9,36] represents another active and important area in the field of fault-tolerant communication. In addition, the meaning of *faults* used in our current discussion also requires clarification. Roughly speaking, we can distinguish between two different classes of faults or failures in communication resources:

- *Soft* failure — This is a *transient* error that occurs randomly and is mostly *noise* related. A typical example would be the garbling of a few bits during the transmission of data across the physical channels.

- *Hard* failure — This is a *persistent* error, whose occurrence may also be random; once it occurs, the error condition will persist until the faulty unit is physically attended to. An example would be an irreversible failure such as a damaged circuit component.

For our purpose, the most important distinction between these two classes of failures is that while the impact of soft errors can be minimized through a combination of temporal and hardware (*ie*, physical resources) redundancies, the impact of hard errors can only be minimized by providing hardware redundancies. An interesting discussion of some high-level techniques and strategies for handling various types of soft failures in the message-communication network of multicomputers can be found in [3]. The success of many of these techniques, however, depends on the continuous operation of the physical communication network. Unfortunately, for the oblivious wormhole networks, even a single broken channel has the effect of disconnecting many source–destination pairs. Obviously, having the ability to route and deliver messages among the subset of nodes that *survived*, in spite of a limited number of hard failures in the message network, is

a *sine qua non* to supporting truly fault-tolerant computation in multicomputers. The research reported in this chapter represents an attempt toward achieving this goal.

4.1.1 The Fault-Tolerant Routing Problem

In order to understand the main problem involved in performing fault-tolerant routing in multicomputer networks, we observe that the popular connection topologies of multicomputer networks such as k-ary-n-cubes or meshes are highly *regular*. Apart from the obvious advantages of having reasonably high bandwidths and systematic layouts, the regularity in these topologies allows for simple *algorithmic*-routing procedures based entirely on local information. In our adaptive-routing formulation, the existence of an algorithmic-routing procedure is essential for keeping the cost of realizing the routing relations at each node at an acceptable level. Such capability is particularly important in fine-grain multicomputers where resources at each node are scarce. Equally important, the existence of simple algorithmic routing procedures in these regular topologies allows direct *hardware* realization of the routing functions, which is absolutely essential in high-performance systems. The *Torus Routing Chip* [11] and the *Mesh Routing Chip* [14] are successful examples of these hardware routers.

As individual nodes and channels fail, the regularity in these networks are destroyed and the algorithmic-routing procedures are no longer applicable. Routing in irregular networks can be systematically achieved by storing and consulting routing tables at each node of the network. However, such routing tables demand excessive resources at each node and become unacceptable as the networks grow in size. Schemes such as hierarchical clustering of network nodes have been proposed [28] to achieve savings in routing-table sizes. However, such methods appear to be unsatisfactory for multicomputer networks for the following reasons:

1. The hierarchical clustering of nodes in the richly connected topologies that are commonly used in multicomputer networks typically eliminates the use of many alternative paths by confining routes to those paths that are consistent with the hierarchy.

2. As nodes and channels fail, the change in the network topology could force a global renaming of network nodes, in the course of adjusting the hierarchy.

3. The circuitry required to store, consult, and update the routing tables represent considerable hardware overhead cost paid in advance in every node, *regardless* of the presence or absence of failures.

Conceivably, another alternative would be to devise some systematic search technique with appropriate backtracking capabilities that would allow the packet to home in on its destination. However, such schemes tend to be difficult and complicated and, in general, are very inefficient in using the very precious remaining bandwidth. Essentially, these approaches are developed to handle routing in highly irregular networks. A different and more satisfactory approach would try to exploit the regularity of the original nonfaulty network. In this chapter, we suggest and investigate such an approach based on our adaptive multipath-routing formulation.

4.1.2 A Simple Fault Model

We now describe a simple fault model to be used in our subsequent discussion of fault-tolerant routing in multicomputer networks. As before, a multicomputer network, M, is a connected, undirected graph, $M = G(N, C)$. The vertices of the graph, N, represent the set of computing nodes; the edges of the graph, C, represent the set of bidirectional communication channels. Recall from Chapter 2 that a computing node is conceptually divided into four subsystems: processor, node memory, message interface, and network router. Since our primary subject of interest is message routing, we shall not make the fine distinction between a node and its router. We now state the additional definitions and notations needed for discussing faulty multicomputer networks:

Definition 4.1 A node (*ie*, its router) $n \in N$ is ***faulty*** if it is unable to perform its packet-to-channel routing-assignment functions correctly.

Definition 4.2 A channel $c \in C$ is ***faulty*** if it is unable to forward packets and follow the coherent protocol correctly.

Definition 4.3 A network is ***faulty*** if it contains at least one faulty node or one faulty channel. The non-faulty nodes and channels are referred to as the ***survived nodes*** and ***survived channels***. Given a faulty network, the set of all survived nodes and survived channels together constitutes the ***survived subnet***.

Definition 4.4 A route p_{ij} from node n_i to node n_j generated by R of the original non-faulty network is *legal* in its faulty descendents if, and only if, both the source, n_i, destination, n_j, and all intermediate nodes and channels of p_{ij} remain ***non-faulty*** in the faulty network. Observe that whether a route remains legal depends on the particular fault pattern of the network under consideration.

We assume the following:

1. A packet forwarded to a faulty node is consumed by that node.
2. A packet forwarded along a faulty channel is lost.
3. A survived node is able to determine the status of all of its own channels locally.
4. Node faults and channel faults occur independently and randomly.

Together, the first three assumptions capture the essence of the well-known notion of a ***fail-stop*** processor that has well-defined failure-mode operating characteristics [46]. In particular, it assumes that a faulty node simply stops executing without performing incorrect routing or generating spurious messages into the network. More specifically, we approach the problem of fault-tolerant routing by assuming the existence of a multicomputer network consisting of such fail-stop nodes and channels in order to motivate the issues, and discuss the solutions at an abstract level.

The independent and random occurrences of node faults and channel faults model an environment of random component failures such as malfunctioning integrated circuits or failed connections due to bad contacts. Fault distributions that are highly correlated and non-random in character will require a different approach from what is explored

here. A very good example of such non-random faults is found in power-supply failures. In a typical arrangement, a single power supply is used to provide power to a whole cluster of nodes, such as a contiguous submesh. Hence a single power-supply failure will disable the entire chunk of nodes supported by it. A natural solution to handle such failures is to provide redundancy at the location of the bottleneck, *ie*, to provide for backup power supplies, instead of relying on the routing network.

It is interesting to observe that under our model, a non-faulty node cannot distinguish a faulty neighboring node from a faulty channel joining them. In fact, a faulty node is empirically equivalent to a survived node whose channels are all faulty. In a limited way, the notion of a faulty node models a highly correlated local concentration of channel faults.

4.2 Systematic Fault-Tolerant Routing

Motivated by our desire to build high performance networks through hardware realization of the routing operations, we look for the solution that will allow us to continue to use, with minimal change, the original routing hardware for the non-faulty network, so as to exploit the inherent regularity in these multicomputer networks. To proceed, observe that an immediate result of having only local information to guide routing is that pairs of survived nodes may not be able to communicate with each other even if they remain connected. In this section, we introduce and define two theoretical notions that characterize the situations under which we can continue to use the algorithmic routing relations, defined for the original non-faulty networks, to systematically direct routing in its faulty descendents.

One immediate advantage of this insistence in using only the set of original routes is that we can obtain *a priori* bounds on the length of routes joining pairs of sources and destinations in the survived subnet. Another advantage, as we shall see, is that it is very easy to detect messages destined to faulty nodes based only on locally available information. Once detected, such messages can then be sunk into an intermediate node, thus triggering exception processing.

Obviously, a scheme that confines the routing paths to those used in the original non-faulty networks would fail miserably if the routing functions are oblivious. The effectiveness of this scheme under our adaptive routing formulation will ultimately depend on the connection topology and on the set of routing relations defined by the algorithmic-routing procedure.

4.2.1 The Convex Subset

We shall first introduce the notion informally using the 2D rectilinear mesh network that employs the usual shortest-path routing relation as an illustration. Consider Figure 4.1, which depicts a 2D mesh network whose faulty nodes are disconnected from the bulk of the survived nodes. It is straightforward to observe that the illustrated fault pattern has the following interesting property: There exists at least one legal route between every pair of survived nodes. Recall that a route of the original network is legal in its faulty descendents if the route lies completely within the set of survived nodes and channels. This property is ***sufficient*** to allow all survived nodes to communicate by sending and

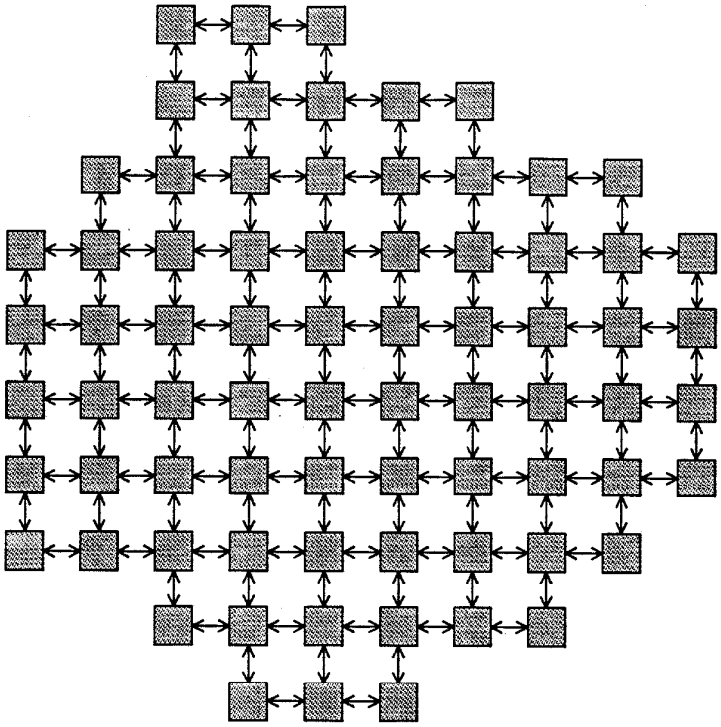

Figure 4.1: A Convex Survived Set in a 2D Mesh Network

receiving messages among one another according to the routing relations of the original non-faulty network. In particular, given a survived node, n_i, and its corresponding set of survived channels, $C_i' \subseteq C_i$, let $R_{ij}' \subseteq C_i'$ denote the *restricted* routing relation of n_i to another node, n_j; *ie*, $R_{ij}' \equiv R_{ij} \cap C_i'$. The above sufficiency property is equivalent to having $R_{ij}' \neq \emptyset$ for every pair of survived nodes n_i and n_j. Under this condition, each message packet that arrives at an intermediate node but is destined to another survived node will find from the restricted routing relation of the intermediate node at least one profitable channel that is non-faulty. The acyclicity of the original routing relations then guarantees the existence of a route. In summary, these observations motivate the following definition:

Definition 4.5 Given the set $\mathcal{R} = \{R_{ij}\}$ of routing relations of a non-faulty network, a set of survived nodes, $S \subseteq N$, is *convex* under $\mathcal{R}$, if for every pair of nodes, $n_i, n_j \in S$, there exists at least one legal route leading from n_i to n_j that is generated by $\mathcal{R}$.

Notice that the original non-faulty network is convex by definition. When the entire set of survived nodes and channels of our network forms a convex set, we have the happy situation where all survived nodes can continue to communicate with each other. These communications are achieved with virtually no change in the basic routing decision mechanism at each node. The only added requirement is for a router to be able to recognize its own faulty channels. This property renders a direct hardware realization practical.

When the set of survived nodes and channels does not form a convex set under $\mathcal{R}$, because certain pairs of nodes do not have legal routes joining them that are consistent

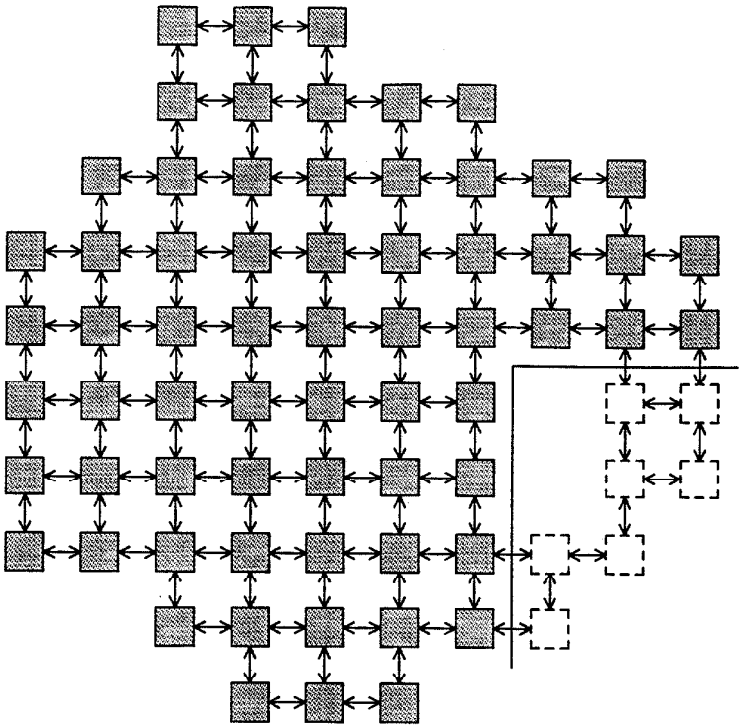

Figure 4.2: A Convex Survived *Subset* in a 2D Mesh Network

with $\mathcal{R}$, we have a problem. In the interest of pursuing simplicity and generality, we suggest and investigate the following simple alternative: to selectively discard certain survived nodes that are particularly difficult to communicate with, so that the remaining subset of survived nodes constitutes a convex set. In other words, instead of asking how to route messages in an irregular network, we ask the alternative question: how to *restore regularity*, or in this case, convexity, back to the survived network. In essence, nodes that become difficult to reach without global information are abandoned as a result of our insistence on using only local information to guide routing. This approach immediately suggests the following problem.

Problem 4.1 (Maximum Convex Subset) Given the set $\mathcal{R}$ of routing relations of a non-faulty network, the set of survived nodes $S \subseteq N$, and survived channels, find the maximum cardinality subset $S_C \subseteq S$, which is convex under $\mathcal{R}$.

By restricting all computations and communications to within the convex subset of a survived network, routing of messages can again be carried out by the hardware router implementing the original set $\mathcal{R}$ of routing relations, a very simple state of affairs. An illustration of this technique is depicted in Figure 4.2, where the broken nodes denote nodes that are deliberately disconnected from the bulk of the remaining network. The cardinality of the maximum convex subset of a survived network provides a useful figure of merit to gauge the effectiveness of this approach to fault-tolerant routing.

4.2.2 The Communication Kernel

The finding of the Maximum Convex Subset of the set of survived nodes is one useful simple strategy that enables us to regularize the survived network and, hence, to be able to continue to use a restricted version of the original routing relations to guide message routing. We now describe another useful regularization strategy, namely, restraining certain survived nodes to operate only as pure *switches*: A switch can only forward messages but *cannot* itself generate and consume messages. In particular, a switch can never be the destination of a message. Instead, its presence is to retain certain routes in order to enable communication between pairs of nodes that would otherwise be impossible. The rationale is that some survived nodes that are difficult to reach, and thus are discardable, might be located in positions that enable other pairs to communicate and, thus, should be retained. Restraining such nodes to act as pure switches represents an attempt to capture both conflicting objectives. Because of the nondeterministic nature of message trajectories in our adaptive formulation, a node is eligible to be a message destination if and only if legal routes that are generated by $\mathcal{R}$ exist between it and every survived node. This way, a message with sufficiently high priority will always be forwarded toward its destination, regardless of where the message is. These observations motivate the following definition:

Definition 4.6 Given the set $\mathcal{R} = \{R_{ij}\}$ of routing relations of a non-faulty network, and $S \subseteq N$, the set of survived nodes, the set $K \subseteq S$, called the *communication kernel* of S under $\mathcal{R}$, is the set of survived nodes $\{n_j\}$, where for all $n_i \in S$, at least one legal route that is generated by $\mathcal{R}$ exists that leads from n_i to n_j.

Notice that the kernel of a non-faulty network is by definition the entire network. Given a fixed nonempty set of survived nodes, its communication kernel always exists and is unique. The word *kernel* is jargon borrowed from computational geometry, where the kernel of a simple polygon is defined to be the set of points inside the polygon that are visible from every point in the polygon. The cardinality of the communication kernel of a survived network provides a useful figure of merit to gauge the effectiveness of this strategy.

These two regularization strategies, namely, selectively discarding nodes, and selectively restraining certain nodes to operate as pure switches, can be combined to achieve even better node-reclamation results. By deliberately discarding certain nodes that are excessively difficult to reach, it may be possible to increase the cardinality of the communication kernel of the remaining nodes. This observation suggests the following problem.

Problem 4.2 (Maximum Communication Kernel) Given the set $\mathcal{R}$ of routing relations of a non-faulty network, the set of survived nodes $S \subseteq N$, and the survived channels, find the subset $S_K \subseteq S$ that contains a communication kernel of maximum cardinality, over all such subsets, under $\mathcal{R}$.

Figure 4.3 shows a typical communication kernel obtained for a 2D faulty mesh network. The blank nodes in the figure are nodes that have been restrained to operate as pure switches. The ratio of this maximum kernel cardinality to the size of the original non-faulty network serves as a natural reliability measure for our subsequent investigation.

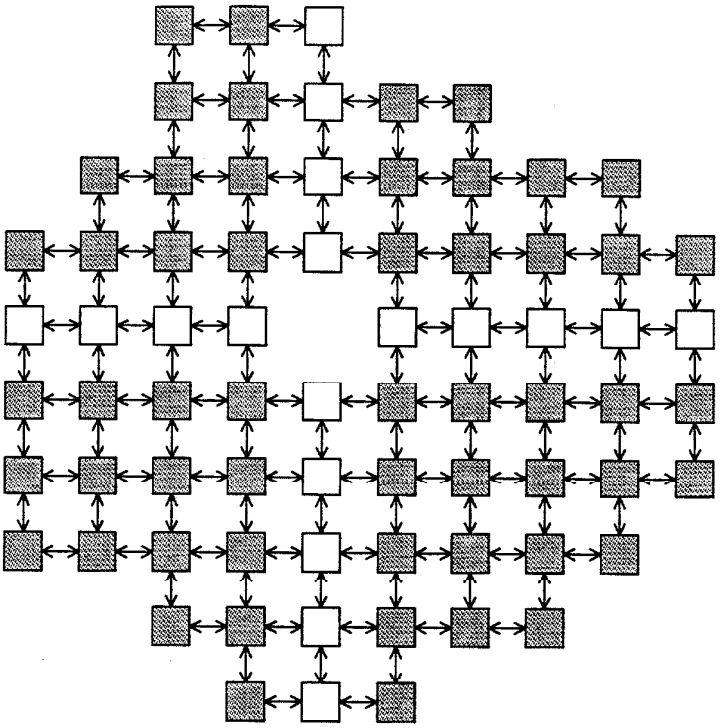

Figure 4.3: An Example Communication Kernel in the 2D Mesh

4.3 Computational Considerations

In the previous section we have motivated and defined the two problems — MCS, finding the maximum convex subset, and MCK, finding the maximum communication kernel — as our principal regularization strategies that allow for the continuing use of $\mathcal{R}$ to guide message routing in a faulty network. Their computations are the subject of this section.

4.3.1 Computational Complexity

Before one can meaningfully derive effective computational schemes to determine the MCS and MCK for any survived network, it is natural to first investigate the complexity of the involved computations. We shall start by proving the NP-completeness of the two decision problems; this is sufficient to establish the NP-hardness of the two corresponding search problems [16].

Theorem 4.1 The maximum convex subset (MCS) decision problem is NP-complete: Given an arbitrary network, $M = (N,C)$, an arbitrary set of routing relations, an arbitrary set of survived nodes and channels, and a positive integer $J \leq |N|$, determine if M contains a survived convex subset of size J or more.

Proof. MCS, as defined, is in the class NP; *ie*, given a subset of survived nodes, there exists polynomial time algorithms for checking the convexity of the subset. We now transform the *Maximum Clique* decision problem for an arbitrary graph, G, of $|N|$

vertices into MCS. To proceed, we first define our original non-faulty network M to be the completely connected graph, K_n, and routing relations, $\mathcal{R} = \{R_{ij}\}$ with $R_{ij} = \{c_{ij}\}$, where c_{ij} is the only channel joining node n_i to node n_j in $M = K_n$. The transformation from the arbitrary graph, G, to a survived subset of M is defined as follows: Vertex v_i in G is mapped to node n_i in M, and the edge e_{ij} in G is mapped to a survived channel c_{ij} in M. All remaining channels in M are considered to be faulty. Under this mapping, a clique in G is equivalent to a convex subset in M under $\mathcal{R}$. Hence, a solution to the MCS applied to the network M can be trivially transformed back to give a maximum clique for G. This establishes the NP-completeness of MCS. ∎

Theorem 4.2 The maximum communication kernel (MCK) decision problem is NP-complete: Given an arbitrary network, $M = (N, C)$, an arbitrary set of routing relations, an arbitrary set of survived nodes and channels, and a positive integer, $J \leq |N|$, determine if M contains a survived subset having a communication kernel of size J or more.

Proof. The proof is almost identical to that in MCS. The network, M, the set of routing relations, $\mathcal{R}$, and the vertex-to-node and edge-to-channel mappings are all identically defined. The proof is completed by observing that, for the defined network, M, if a survived subset contains a communication kernel of size J or more, it also contains a convex subset of size J or more, and vice versa, by definition. Hence, a solution to the MCK problem in this case can also be transformed back to give a maximum clique for G. This establishes the NP-completeness of MCK. ∎

The above NP-completeness results imply that the corresponding search problems of actually finding the maximum convex subset and maximum communication kernel subset are NP-hard; hence, unless P = NP, exact solutions that run in polynomial time will not be found. The question as to whether the restriction of certain fixed topologies and certain fixed routing relations would reduce the problem to polynomial time remains open. In any case, in the interest of studying the performance of different network topologies and routing relations at the current stage, we shall have to rely on computations that have been done by approximating heuristics.

4.3.2 Approximating Heuristics

We now proceed to describe a simple but effective heuristic elimination procedure to find reasonable solutions to the MCK problem. Whether there exists approximation algorithms capable of providing *a priori* performance guarantees is yet another open question.

The heuristic elimination procedure to be described is motivated naturally by the objectives behind our introduction of the MCK problem. The objectives are:

1. Selectively discard a subset of survived nodes that are difficult to reach given the survived structure.

2. Selectively restrain a subset of survived nodes to operate as pure switches. These switches maintain the reachability among other pairs of nodes without insisting on being reachable themselves.

We observe that the second objective can be readily achieved, once we have decided which subset of survived nodes to discard. This is because every fixed subset of survived nodes has a *unique* communication kernel, by definition. Any survived node that is not inside the kernel will then be restrained to operate only as a switch. On the contrary, achieving the first objective optimally is far more difficult since there are an exponential number of candidate subsets to consider. Here we suggest the following heuristic approach that delivered reasonable results for all our subsequent simulation studies.

> Sequentially discard one node at a time by eliminating the one that is the most difficult to reach at that moment. During the entire elimination process, keep track of the maximum cardinality kernel ever obtained. Terminate when the number of remaining survived nodes is equal to the size of the recorded maximum kernel; at this point, we are sure no further improvement is possible under this heuristic.

In order to apply this heuristic, we need a way to quantify the vague notion of a node being *difficult to reach*. We suggest the following intuitive definition, which appears to give good empirical results:

> Given a survived node, n_j, let $h(n_j)$ denote the total number of survived nodes, $n_i \neq n_j$, for which there is no legal route leading from n_i to n_j. The node, n^*, with the maximum count, *ie*, $h(n^*) \geq h(n_j)$, $\forall j$ is designated as the node that is the most difficult to reach. Ties can be broken arbitrarily.

Oftentimes, the computation of the cost estimate, $h(n_j)$, can be rather expensive; in such case, a simpler estimate, $h'(n_j)$, which counts the number of other survived nodes, n_i for which $R'_{ij} = \emptyset$, may be used instead. The simpler estimate, $h'(n_j)$, shares with the more accurate estimate, $h(n_j)$, the property that at each stage of the computation, all nodes in the communication kernel have their respective cost estimates equal to zero. Thus, computing the suggested cost estimates after each elimination also simultaneously computes the desired new kernel configuration. Furthermore, since the elimination procedure evolves sequentially until it is *suspended* when the desired maximum kernel has been found, it is possible to *resume* and continue the elimination procedure when new faulty nodes or channels are identified. In other words, it is conceivable that we can run the elimination procedure as a background process in each node that serves as an exception handler, to be activated when additional faulty nodes or channels are detected.

An implementation of this heuristic running on the iPSC1/d7 Cubes [23] provides the principal tool for our simulation studies of the effectiveness of this approach to fault-tolerant routing in multicomputer networks.

4.4 Simulation Experiments and Results

In this section, we present the simulation and computation results for the important class of n-dimensional rectilinear meshes. We shall abuse the nomenclature somewhat by referring to them as the k-ary-n-meshes, although these networks do not have end-around connections. One major advantage of these networks is the existence of very

simple routing relations that are based on the natural *city-block* or L_1-metric defined over these n-dimensional grids. For any two points, $p_1 = (a_1, a_2, \cdots, a_n)$ and $p_2 = (b_1, b_2, \cdots, b_n)$, the L_1 metric $d_{L_1}(p_1, p_2)$ is defined as follows:

$$d_{L_1}(p_1, p_2) = \sum_{i=1}^{n} |a_i - b_i|$$

Notice that when $k = 2$, we have the familiar binary-n-cube, and when $n = 2$, we have the $k \times k$ 2D mesh. Our primary figure of merit in these simulations is defined as follows:

Definition 4.7 The *yield* is the fraction of operating nodes reclaimed by the kernel computation, as compared to that of the total number of nodes in the original non-faulty network.

Notice that even in a *perfect* scenario, the yield would be identical to the fraction of survived nodes, which is always < 1, in any faulty network. In a crude sense, it represents a first-order measure of the degradation in computing power due to the ensuing faults. Specifically, our goals in performing these simulation studies are:

1. To collect the empirical statistics on the effectiveness of our approach for providing fault-tolerant routing to a popular class of networks that has practical importance.

2. To quantitatively compare the tradeoff of arity versus dimension under the random- and independent-fault assumption.

3. To assess and interpret the empirical behavior of these tradeoffs in order to provide insights into suggesting means for improvement.

To help differentiate the impact of node failures and channel failures on the effectiveness of our approach, we simulated separately the two distinct cases of having purely node faults and having purely channel faults. The faults are generated independently using identical probabilities chosen over a range from one to ten percent. Figures 4.4 to 4.9 plot the simulation and computation results obtained for three different networks: the 2-ary-10-cube, the 4-ary-5-mesh, and the 32-ary-2-mesh. Notice that all three networks were chosen to have 1024 nodes, representing the range from *low-* through *medium-* to *high*-dimensional networks. In these figures, we have chosen the scattered plots in order to convey pictorially the statistical distributions of the collected simulation results. From these plots we notice the following general trends:

1. In all three networks, the yield of the purely channel failure case is worse than the corresponding yield of the purely node failure case of the same failure percentage. That such should be the case is not straightforward, since each node that is faulty renders all its incident channels faulty. In fact, for our chosen range of fault probability, under the same percentage of faults, the total number of channels that are rendered faulty due to purely random node faults would be approximately twice that of the corresponding number due to purely random channel faults. Apparently, the very dispersive occurrences of pure channel faults is much more effective in destroying routes in these survived networks.

2. The yield obtained under both the random node faults and random channel faults decreases as we decrease the dimension of the networks. This fact is consistent with our intuition, since the average number of distinct paths between source and destination is a combinatorial function that increases steeply with the dimension of the networks. In fact, the differences are so great that, from the empirical figures, one is justified in concluding that the binary-n-cube is extremely robust under our adaptive scheme, whereas the 2D mesh, at least in its present form, is not.

3. The fitted curve is a much better representation of the actual statistical distribution of simulation results in the higher-dimensional networks than the lower-dimensional ones. The lower-dimensional networks have distributions that are highly dispersive. Again, this fact can be understood statistically from the combinatorial disparity in the number of paths across networks with different dimensions. A larger number of distinct paths implies a better convergence to the mean value according to the law of large numbers.

While the above empirical results indicate that the binary-n-cube network is much more robust than the 2D mesh connection, this advantage is offset by the excessive wire bisection required to connect this topology. A lucid explanation of the wire bisection argument can be found in [13]. Perhaps most importantly, low-dimensional networks such as 2D or 3D meshes or tori are practical in that they are readily realizable. In particular, 2D structures are very desirable since they are much cheaper to build. Furthermore, they leave the third physical dimension behind; this is extremely convenient for service and maintenance purposes that are particularly important in large networks. However, the above empirical results for the 2D mesh connection are rather disappointing. It is therefore desirable to look for methods to improve the yield while remaining in 2D. Such a network will be presented in the next section. To proceed, let us first examine more closely the reason behind the poor performance of the 2D rectilinear mesh.

Figure 4.12 shows a typical kernel obtained in a 2D mesh network when there are only a few faults. In this figure, the solid boxes denote reclaimed nodes, the blank boxes denote switching nodes, the broken boxes denote discarded nodes, and the missing nodes and channels are faulty. Observe that, except for the possibility of discarding whole corner regions, each faulty channel has to be accommodated by eliminating either an entire row *or* an entire column of nodes, whereas, node fault causes both an entire row *and* an entire column to be eliminated. It is straightforward to see that it does not take many faulty nodes or channels before the mesh completely collapses. The problem exists because although our adaptive routing scheme allows us to exploit multiple alternate routes, if such exist, the routing relations defined by the city block metric leave many source-destination pairs with only a *unique* route between them. In particular, any pair of nodes that lies on the same row or column is connected via a single route defined under the city block metric. In essence, the set of routes defined by the routing relation for the 2D mesh topology simply is not rich enough in connectivity. It is clear that in order to improve the reliability, such bottlenecks must be removed.

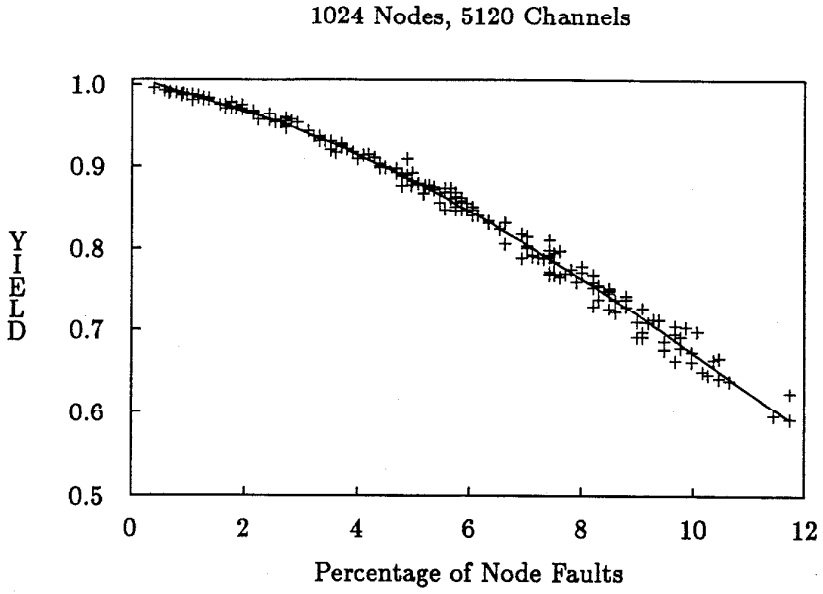

Figure 4.4: Binary-10-Cube with Node Faults

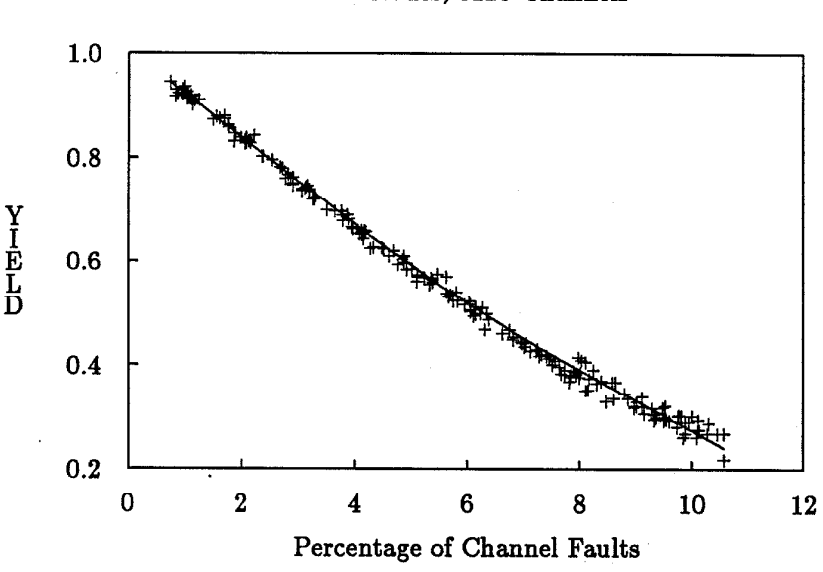

Figure 4.5: Binary-10-Cube with Channel Faults

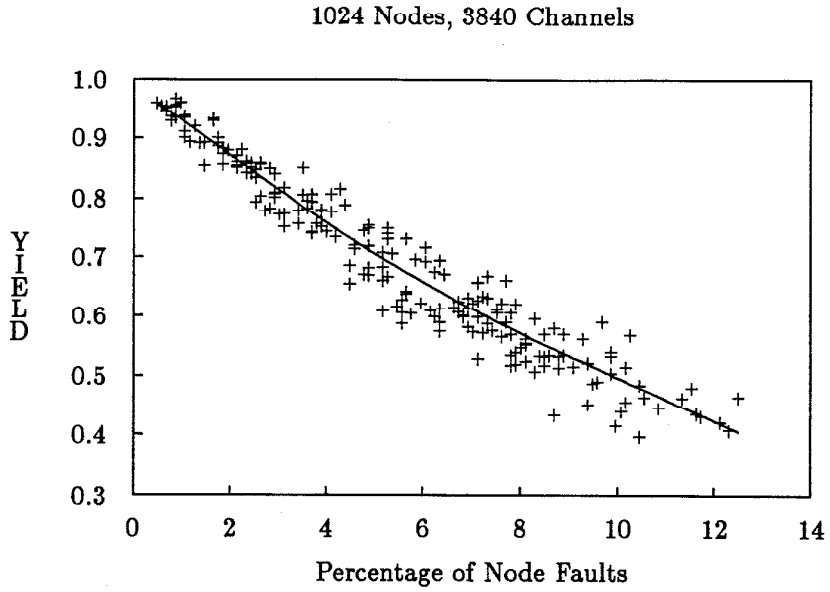

Figure 4.6: 4-Ary-5-Mesh with Node Faults

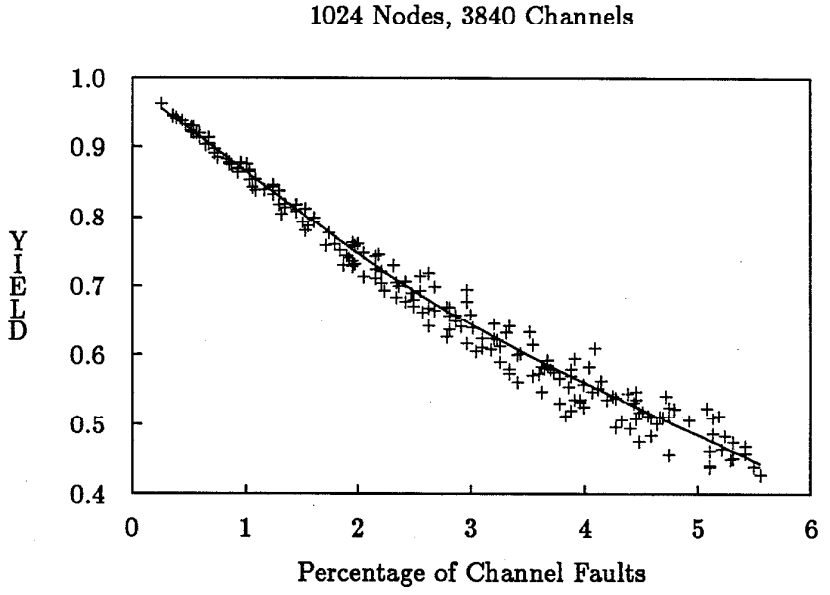

Figure 4.7: 4-Ary-5-Mesh with Channel Faults

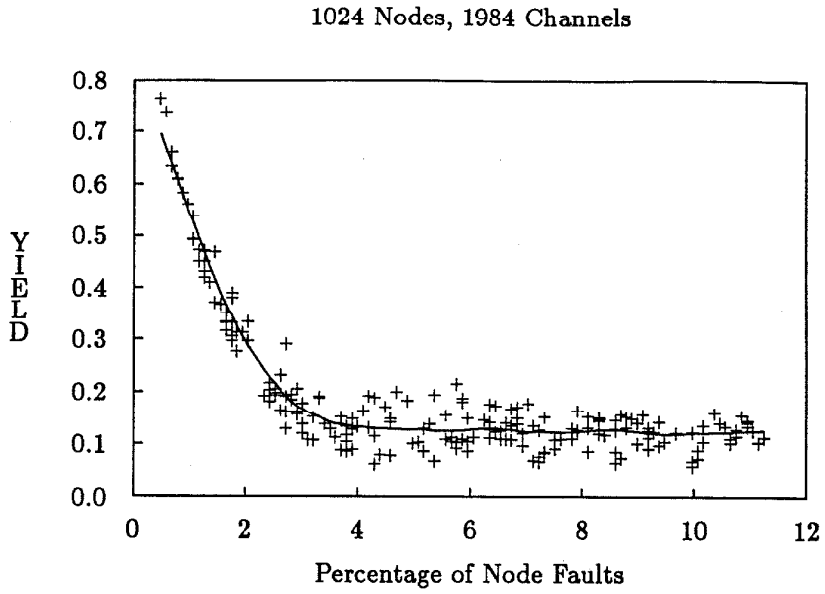

Figure 4.8: 32 × 32 Rectilinear Mesh with Node Faults

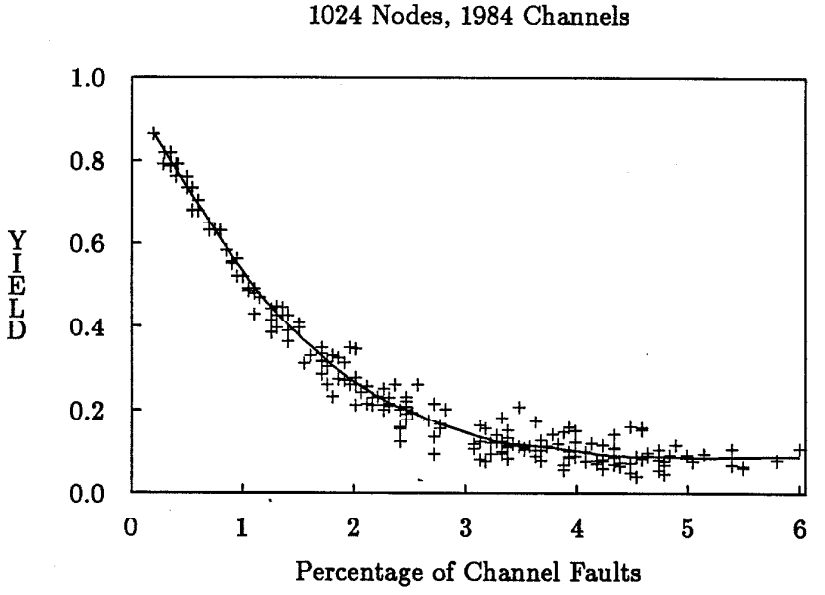

Figure 4.9: 32 × 32 Rectilinear Mesh with Channel Faults

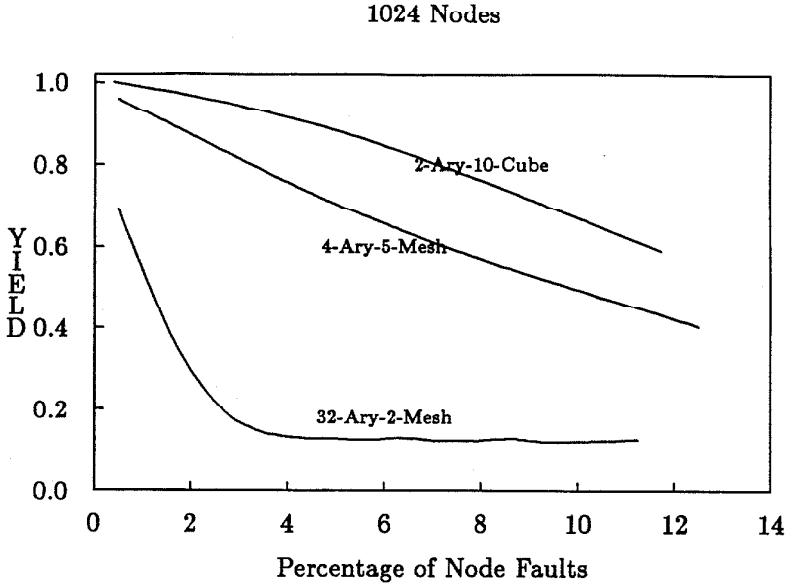

Figure 4.10: A Comparison of Yield with Node Faults

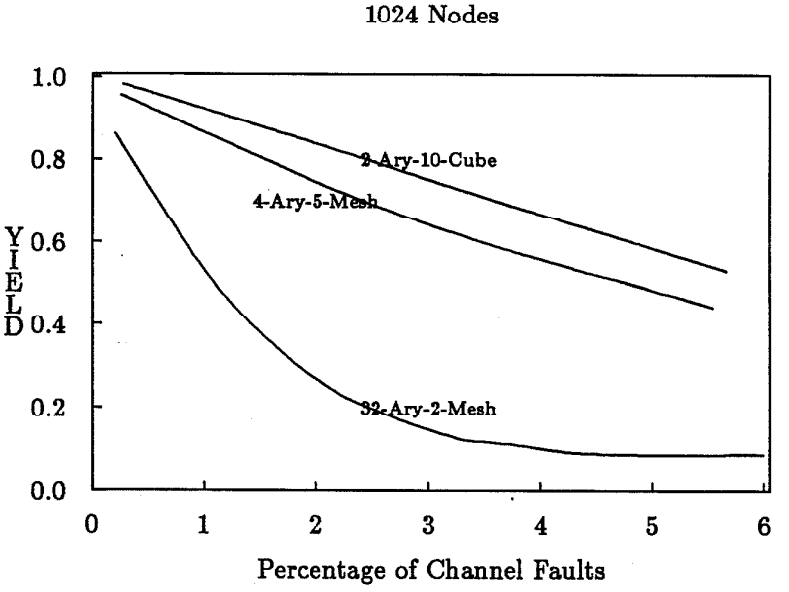

Figure 4.11: A Comparison of Yield with Channel Faults

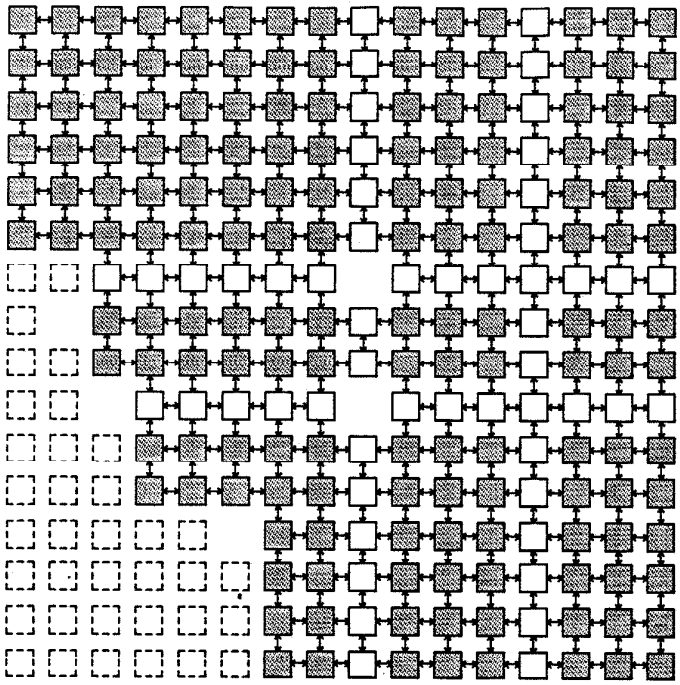

Figure 4.12: A Typical Kernel for the 2D Mesh Network

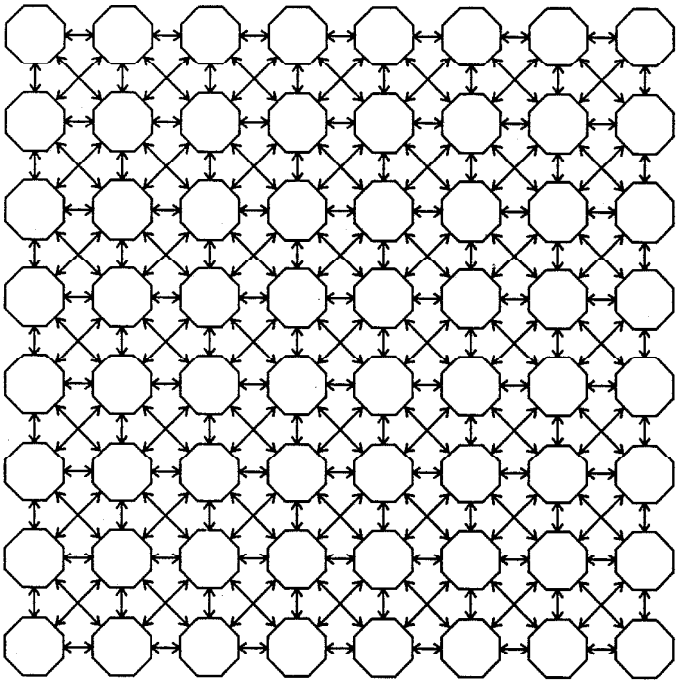

Figure 4.13: The Octagonal Mesh Network

4.5 The Octagonal Mesh Network

In the previous section we have presented evidence that the existing routing relations, which are based on the natural shortest graph-theoretic distance or L_1-metric defined over n-dimensional grids, are too weak for lower-dimensional meshes. We have argued that this weakness stems from the fact that with the L_1-metric based routing relations, there are far too many pairs of nodes connected by single routes. Since our approach to fault-tolerance relies heavily on the existence of multiple paths joining pairs of nodes, hindsight clearly reveals that the scheme would fail miserably for these lower-dimensional meshes. In order to increase the reliabilities of these lower-dimensional meshes, we shall need to augment the network with more channels, and to seek a better set of routing relations that will generate a much more richly connected set of routes. In particular, we focus our attention on 2D networks, since they constitute the simplest nontrivial interconnection patterns that are still much cheaper and easier to assemble physically than three- or higher-dimensional structures.

Figure 4.13 depicts a two-dimensional *octagonally* connected mesh network, where each node is connected to eight neighbors. Its channel connections are almost identical to the ordinary 2D rectilinear mesh, except that it is augmented with additional diagonally connected channels. The octagonal mesh shares most of the same advantages with the simpler rectilinear mesh, such as having a systematic, regular layout, and *uniformly short* connections. Furthermore, as we shall see next, it is possible to define a simple set of routing relations that will give rise to a very richly connected set of routes.

4.5.1 The Routing Relation

Given any network, the standard way to generate a set of acceptable routing relations is to define them in terms of the reduction of the standard graph-theoretic distance metric. Specifically, the set of profitable channels for a message is the subset of channels whereby forwarding the message across such a channel will decrease its graph-theoretic distance from destination. For the octagonal mesh, the graph-theoretic distance metric happens to be identical to the L_∞-metric defined as follows:

$$d(p_1, p_2) = d_{L_\infty}(p_1, p_2) = \max(|x_1 - x_2|, |y_1 - y_2|), \qquad \forall\, p_1 = (x_1, y_1),\ p_2 = (x_2, y_2)$$

Under the L_∞-metric, nodes lying along the same row or the same column of the mesh are connected by a ***multiple*** number of routes, provided that they are not adjacent to each other. In other words, using the new octagonal mesh and the L_∞-metric based routing relations, we have succeeded in removing the bottlenecks in our original 2D rectilinear mesh. Unfortunately, in so doing, we have created a different set of bottlenecks: Nodes lying along the same diagonals are now joined only by single routes. Clearly, we need to introduce extra alternate routes in addition to those allowed under the L_∞-metric, provided that, collectively all the generated routes remain *acyclic*.

To proceed, we observe that the two L_1- and L_∞-metrics defined on a 2D mesh are *compatible* with each other over the octagonal mesh. They are compatible in the sense that moving in a direction that *decreases* one metric will *never* result in an *increase* in the other metric. This is summarized in the following lemma:

Lemma 4.3 The L_1- and L_∞-metrics are compatible with each other over the 2D octagonal mesh.

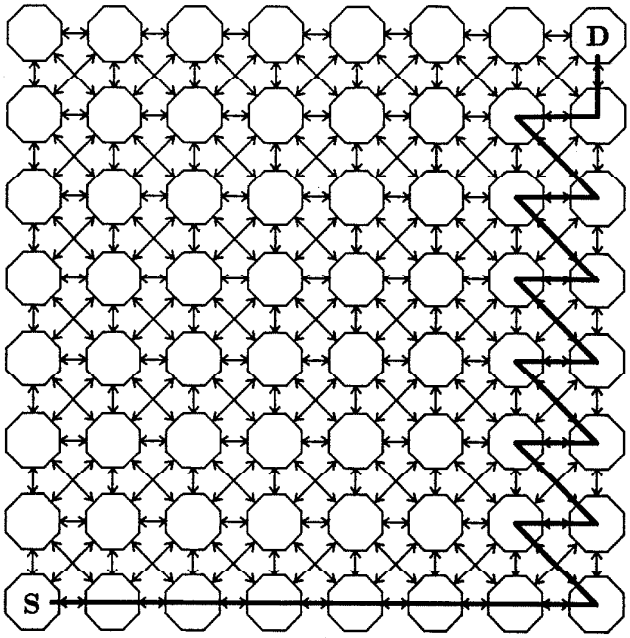

Figure 4.14: A Worst Possible Route in the Octagonal Mesh

Proof. Let δX and δY denote the corresponding change in absolute distances from the destination in the X and Y directions when a message is forwarded across a channel. We observe that during every move, δX and δY can only assume the discrete values from the set $\{-1, 0, +1\}$. To decrease the L_1-metric, we must have $\delta X + \delta Y < 0 \Rightarrow \delta X \leq 0 \wedge \delta Y \leq 0$, which can never increase the L_∞-metric. Similarly, to decrease the L_∞-metric, we must have $\delta X < 0 \vee \delta Y < 0 \Rightarrow \delta X + \delta Y \leq 0$. In other words, it can never increase the L_1-metric. ∎

Compatibility of the L_1- and L_∞- metric allows us to define a new metric M as the sum of the two old metrics:

$$d_M = d_{L_1} + d_{L_\infty}$$

It is straightforward to see that M, defined above, is a genuine metric over the octagonal-mesh network. We shall now define an alternative set of routing relations, $\mathcal{R}^* = \{R^*_{ij}\}$, based on the reduction of this new metric, M:

$$R^*_{ij} = \{c_{ik} \in C_i | c_{ik} \in C_k \wedge d_M(n_k, n_j) < d_M(n_i, n_j)\}$$

which guarantees the acyclicity of routes generated collectively by the R^*s. Notice that another possible way to generate a combined metric for the octagonal mesh is to define it *lexicographically* as (d_{L_∞}, d_{L_1}). We favor the summation-generated metric because its values are more closely related to actual distances. Under this combined metric, the routes generated are no longer shortest graph-theoretic distance paths. However, since the routes are generated so as to reduce at least one of the component metrics of M, and d_{L_1} can be twice as much as d_{L_∞}, it is clear that the route lengths cannot be more than three times those of the shortest distance paths. In fact, on the octagonal

mesh, the worst-case route length is at most $3d_{L_1} - 1$, since the last move on any route must simultaneously reduce both d_{L_1} and d_{L_∞}. That such a worst case does exist is exemplified in Figure 4.14. The advantage we gain from paying such a price is that between every pair of *nonadjacent* nodes, there now exists at least two node-disjoint routes joining them. Hence, with this new set of routing relations, we succeed in removing the bottlenecks that have caused the low yield results in the two-dimensional rectilinear mesh.

4.5.2 Reliability Assessment

In this section, we present the simulation and computation results for the octagonally connected mesh network discussed in the previous section. Again, we have attempted to differentiate the effects of node failure and channel failure by simulating them separately. The results are plotted in Figures 4.15–4.18. The followings observations are made regarding these figures:

- The average yield statistics of the 2D octagonal mesh are far better than the corresponding statistics of the 2D rectilinear mesh under identical failure probabilities.
- The statistical patterns are comparatively more dispersive than those obtained from all the previous simulations.
- The statistical patterns for the node faults are much more dispersive than those obtained from the channel faults, with a number of occasional outlying points.

The relatively dispersive nature of the yield statistics and, in particular, the existence of these outlying points require some further explanation; this also helps us to gain additional insights into how the octagonal mesh achieves the much better yield over that of the rectilinear mesh. In Figures 4.21 and 4.22, we have shown the computed kernel configuration of a few scattered isolated faults, and that of a single cluster of faults. Notice that as long as the fault cluster sizes are small, they can be readily *routed around* by the redundant paths generated under the routing relations, R^*s. However, as the fault cluster sizes increase further, they can no longer be accommodated, and a phenomenon so familiar in the original rectilinear mesh commences: Entire columns, rows, or diagonals are restrained to operate as pure switches. For randomly generated faults, such large fault clusters remain relatively rare until the failure probabilities become relatively high. On the other hand, the occasional occurrences of the few large fault clusters result in the existence of the observed outlying points. In particular, the formation of clusters are much more likely if the failure probabilities of neighboring resources are highly correlated. Since node faults are in effect highly correlated channel faults, this explains why many more outlying points are observed under the independent node faults assumption than are observed under the independent channel faults assumption.

Figures 4.19 and 4.20 compare the yield statistics of the octagonal mesh, the rectilinear mesh, and the binary-n-cube. It is interesting to observe that the yield statistics for the 2D octagonal mesh is consistently *better* than those obtained for the binary-n-cube of the same size. However, it should be pointed out that the significant gain in yield is achieved at the expense of reducing the effective available network bandwidth, as we shall find out in the following subsection.

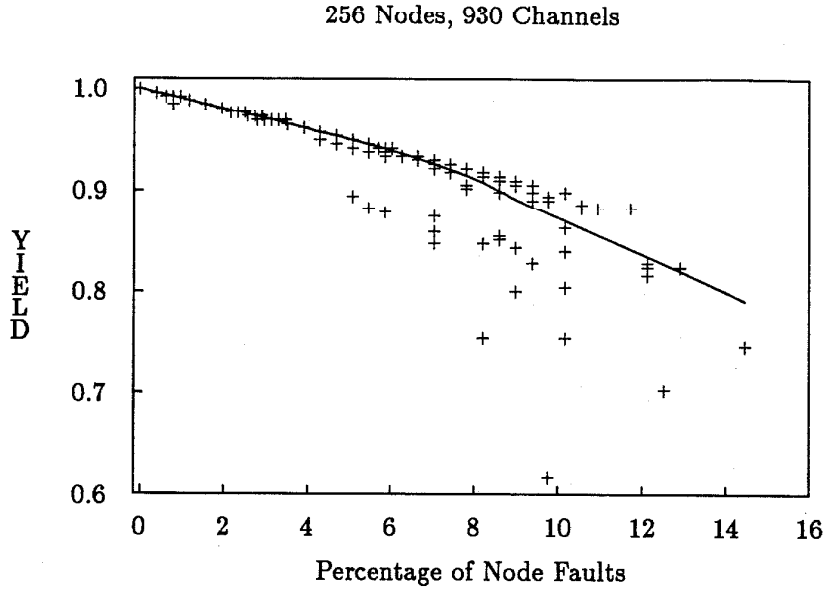

Figure 4.15: 16 × 16 Octagonal Mesh with Node Faults

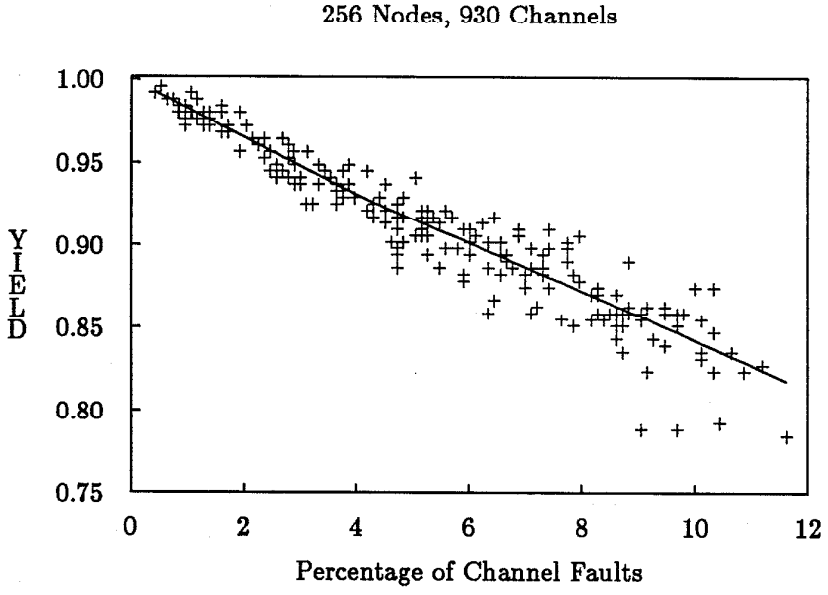

Figure 4.16: 16 × 16 Octagonal Mesh with Channel Faults

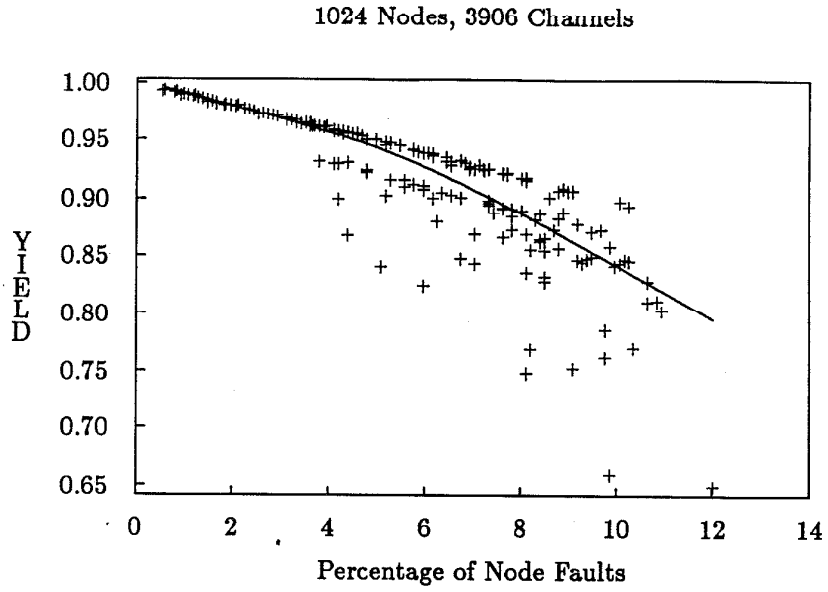

Figure 4.17: 32 × 32 Octagonal Mesh with Node Faults

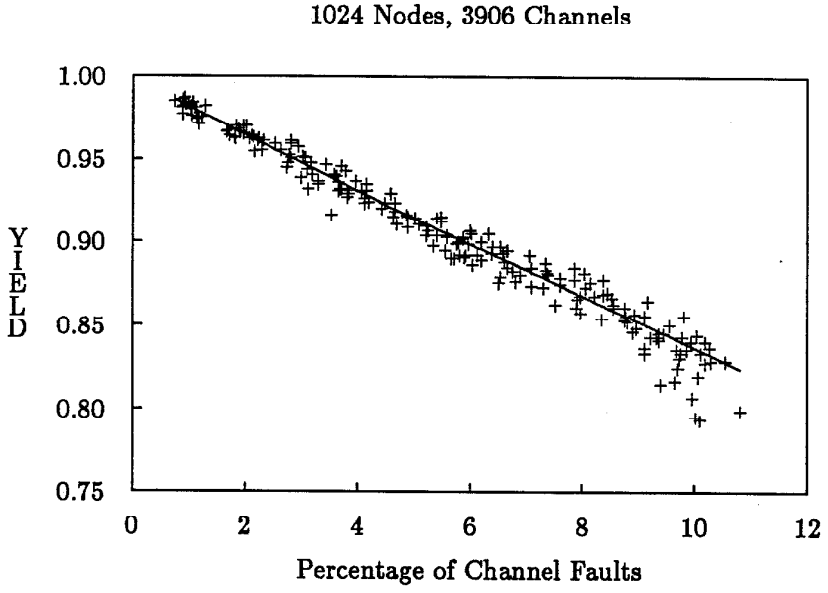

Figure 4.18: 32 × 32 Octagonal Mesh with Channel Faults

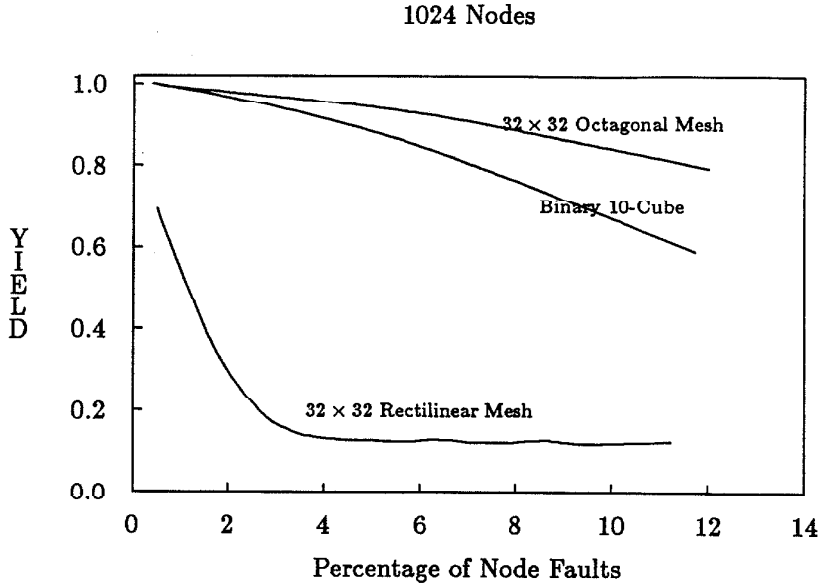

Figure 4.19: Another Comparison of Yield with Node Faults

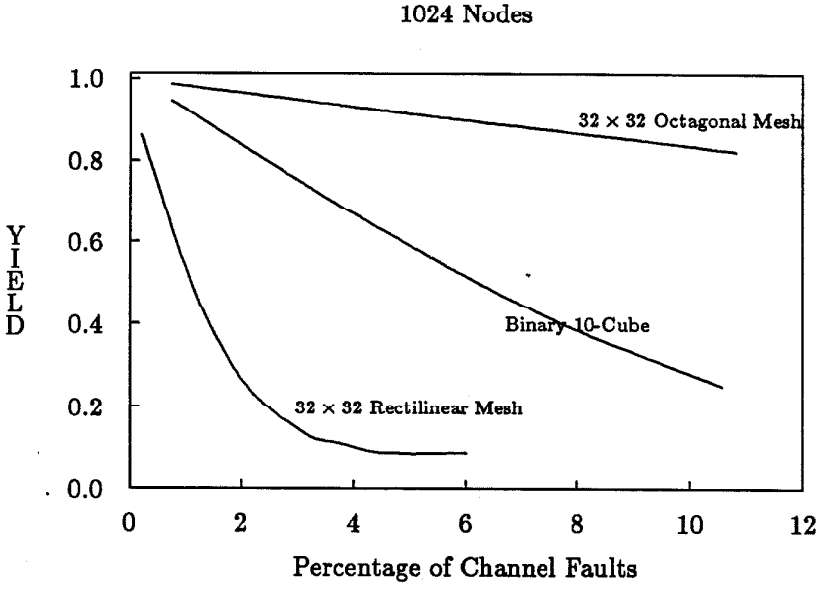

Figure 4.20: Another Comparison of Yield with Channel Faults

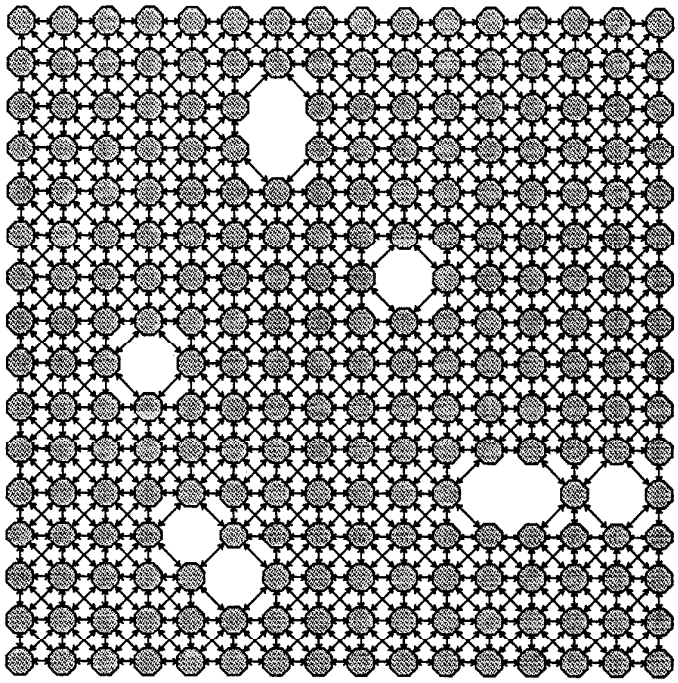

Figure 4.21: A Kernel Configuration Induced by Isolated Faults

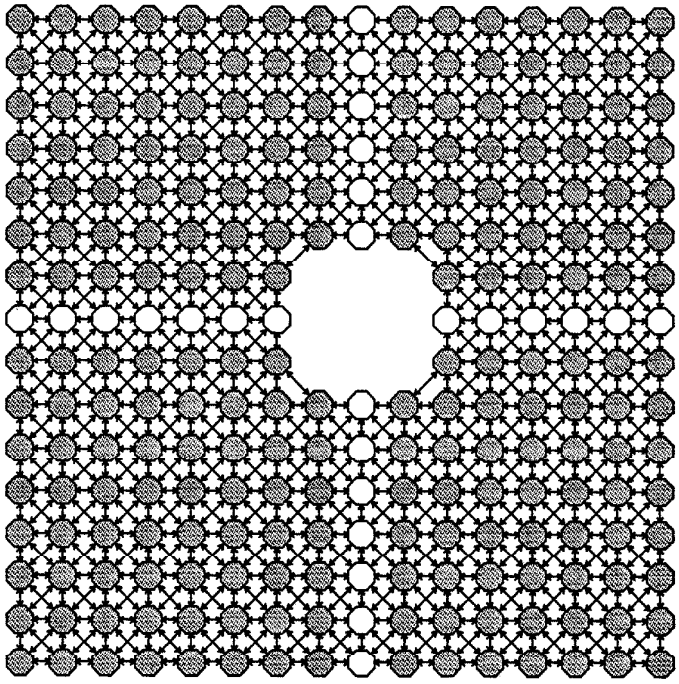

Figure 4.22: A Kernel Configuration Induced by a Cluster of Faults

4.5.3 Performance Assessment

The previous subsection focused on a reliability assessment of the proposed octagonal-mesh network. In this section, we move on to study the various performance aspects of the octagonal mesh. More specifically, our objectives are two-fold:

1. To evaluate the average latency and throughput performances of the octagonal mesh as a potential connection topology in itself; and

2. To obtain preliminary understanding about the amount of communication performance degradation in the presence of random node and channel faults.

Our investigation of the performance behaviors is patterned after that in Chapter 3.

Network Performance Bounds

To start, we shall first derive the bounds on average performance for uniformly random and independent traffic over the $N = k \times k$ octagonal mesh. The argument is similar to that presented in Chapter 3 and is based on the notion of network bisection bandwidth, and the average network distance.

For an $k \times k$ mesh, the network bisection capacity is $3k - 2$ channels in the middle, splitting the network into two halves. Therefore, the average injection rate, q, at each node for random traffic is given by:

$$\begin{aligned} q\left(\frac{k^2}{2}\right)\left(\frac{1}{2}\right) &\leq 3k-2 \\ q &\leq \frac{12k-8}{k^2} = \frac{12}{k} + o\left(\frac{1}{k}\right) \end{aligned}$$

In other words, given that the bisection bandwidth capacity is ≈ 3 times that of the rectilinear mesh of equal size, the expected maximum throughput under ideal conditions should also be ≈ 3 times that of the rectilinear mesh.

Next, in order to obtain the lower bound on the average message latency, it is necessary to obtain the average message distance from its respective destination measured over the octagonal mesh. For the octagonal mesh, the graph-theoretic distance is given by the L_∞-metric, and the average distance, D_{octa}, is defined as:

$$D_{\text{octa}} = \sum_{i=1}^{k-1} iP(\max(\Delta X, \Delta Y) = i)$$

Since for uniformly random and independent traffic, ΔX and ΔY are independent of each other, this can be written equivalently as follows:

$$D_{\text{octa}} = \sum_{i=1}^{k-1} i(P(\Delta X = i)P(\Delta Y \leq i) + P(\Delta X \leq i)P(\Delta Y = i) - P(\Delta X = i)P(\Delta Y = i))$$

In order to evaluate D_{octa}, we need to evaluate the quantities $P(\Delta X = i) = P(\Delta Y = i)$

and $P(\Delta X \leq i) = P(\Delta Y \leq i)$. Following the argument presented in [3], we have:

$$\begin{aligned}
p(\Delta X = 0) &= P(\Delta Y = 0) = \tfrac{1}{k} \\
p(\Delta X = i) &= P(\Delta Y = i) = \tfrac{2(k-i)}{k^2} \qquad 0 < i \leq k-1 \\
P(\Delta X \leq i) &= P(\Delta Y \leq i) = \tfrac{1}{k} + \textstyle\sum_{j=1}^{i} \tfrac{2(k-j)}{k^2} \quad 0 < i \leq k-1 \\
&= \tfrac{k+(2k-1)i-i^2}{k^2}
\end{aligned}$$

From these expressions, we obtain the following, for $0 < i \leq k-1$:

$$\begin{aligned}
P(\max(\Delta X, \Delta Y) = i) &= \left(\tfrac{2\times 2(k-i)}{k^2}\right)\left(\tfrac{k+(2k-1)i-i^2}{k^2}\right) - \left(\tfrac{2(k-i)}{k^2}\right)\left(\tfrac{2(k-i)}{k^2}\right) \\
&= \left(\tfrac{4(k-i)}{k^2}\right)\left(\tfrac{2ki-i^2}{k^2}\right)
\end{aligned}$$

We are now ready to evaluate the average distance from destination, for a reasonable mesh size, k, after simplifying and rearranging terms as follows:

$$\begin{aligned}
D_{\text{octa}} &= \sum_{i=1}^{k-1} \frac{4}{k^4}(2k^2i^2 - 3ki^3 + i^4) \\
&\approx \frac{4}{k^4}\int_{x=0}^{k} (2k^2x^2 - 3kx^3 + x^4)dx \\
&= \left(\frac{8}{k^2}\right)\left(\frac{k^3}{3}\right) - \left(\frac{12}{k^3}\right)\left(\frac{k^4}{4}\right) + \left(\frac{4}{k^4}\right)\left(\frac{k^5}{5}\right) \\
&= \frac{8k}{3} - 3k + \frac{4k}{5} \\
D_{\text{octa}} &\approx \frac{7}{15}k
\end{aligned}$$

For a 32×32 octagonal mesh, the average message distance is ≈ 14.93. This result is very close to that obtained from *Monte Carlo* simulations. Notice that the asymptotic limit is very close to that obtained from a 2D rectilinear torus of the same size.

The Simulation Experiments and Results

In order to obtain insights on the effects of random faults upon the communication performances of the proposed octagonal mesh, three identical sets of simulation experiments were performed on three 16×16 octagonal meshes, each corresponding to a varying degree of faults. The first network has no fault, *ie*, all of its 256 nodes and 930 channels are operational. The second network has a total of 235 nodes and 891 operational channels, whereas the third has a total of 199 nodes and 836 operational channels (see Figures 4.23 and 4.24). We shall henceforth refer to these reclaimed networks as networks A, B, and C, respectively. The faults in networks B and C were generated independently and uniformly under 5% and 12% channel-failure probabilities. Within each set of experiments, the network traffic was uniformly random and independent, with the applied load being varied to cover the entire range. Except for the use of a

different network topology, all the assumptions described in section 3.4 remain valid for the current simulations. Since our main focus here is the investigation of the inherent network performance figures, only the artificially generated traffic experiments were conducted.

Figures 4.25 and 4.27 plot the average normalized throughput versus normalized applied load for the three different networks. Consider, for example, network A (the non-faulty mesh): The normalized network throughput increases linearly with increasing applied load, until it reaches $\approx 72\%$, after which the throughput remains *stable*, in spite of increasingly heavy applied load. Similar behaviors are observed in networks B and C with corresponding saturation throughput values of ≈ 0.58 and 0.38, respectively. The fact that the saturation throughput for network A occurs around 70% can be understood by looking at the very dispersive nature of the routes generated by the routing relations, R^*. In particular, our previous bandwidth argument counted only messages that *must* cross the bisection from their sources on one side to their destinations on the other side. However, many more routes exist that have both their sources and destinations on the same side, but that nonetheless cross the bisection more than once. These routes were not taken into account and are responsible for the $< 100\%$ observed saturation throughput. In other words, a reduced maximum throughput is the price paid in order to obtain a higher network reliability by adopting a more dispersive routing strategy.

Similar throughput behaviors are observed in the reclaimed networks B and C, but having correspondingly lower saturation values. For network B, with 235 nodes, or a yield of $\approx 92\%$, the normalized saturation throughput is reduced from 0.70 to 0.58, or $\approx 80\%$ that of the non-faulty network A. Similarly, for network C, with 199 nodes or a yield of $\approx 78\%$, the saturation throughput is reduced to 0.38, or $\approx 53\%$ that of network A. While these figures are specific to the fault configurations of the two simulated networks, they are suggestive of the extent of performance degradation induced by random faults in general. In particular, if we assume that both the node and channel resources are degraded to the same extent, we may expect that, on the average, the relation between applied load and available bandwidth will remain unchanged. However, the average length of survived routes are increased due to increased reliance on detouring and, hence, they consume additional bandwidth. As a result, we conjecture that the effective saturation throughput will degrade at a rate faster than that of the node and channel resources. The empirical figures obtained for networks B and C appear to be consistent with this conjecture.

The corresponding message latencies for these networks are shown in Figures 4.26 and 4.28. These curves also present no surprises. Again, we observed the familiar characteristic curve for latency behavior. Each curve starts at latency values very close to the theoretical lower bound under low to moderately heavy applied load, and increases rapidly as the load approaches the respective throughput limit. For example, in Figure 4.26, the transition point throughput for networks A, B, and C are $\approx 0.6, 0.45$, and 0.3, respectively. One way to interpret these results is as follows: For computations that are primarily communication bounded, network C, as a result of the ensuing faults, is reduced to $\approx 0.78 \times \frac{0.3}{0.6} \approx 0.39$ of the raw computing speed of the original non-faulty network A. This superlinear degradation in the overall computing performance will in general be observed for traffic patterns generated under random placement strategies that are very effective in maintaining approximate load balance. Because of its simplic-

ity, such random placement strategy is particularly attractive in faulty networks, since these networks are substantially more irregular than the original non-faulty networks.

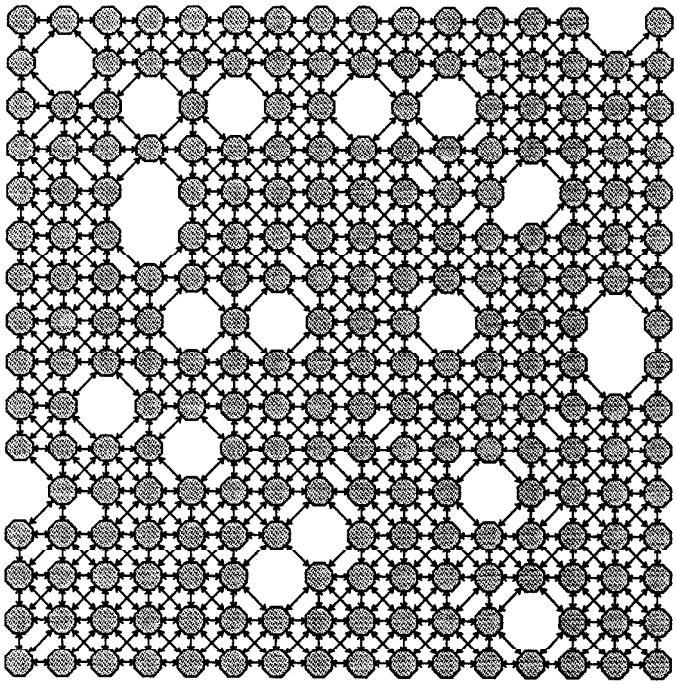

Figure 4.23: Reclaimed Convex Network B — 235 Nodes and 891 Channels

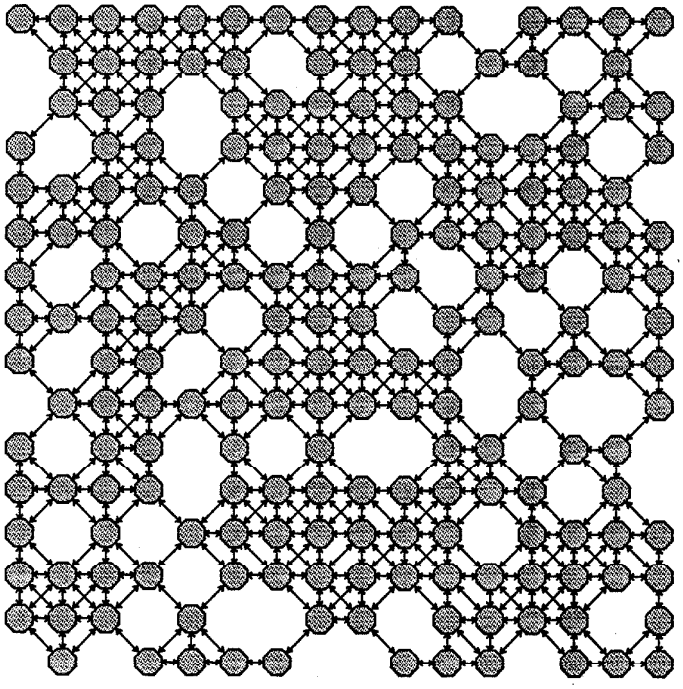

Figure 4.24: Reclaimed Convex Network C — 199 Nodes and 836 Channels

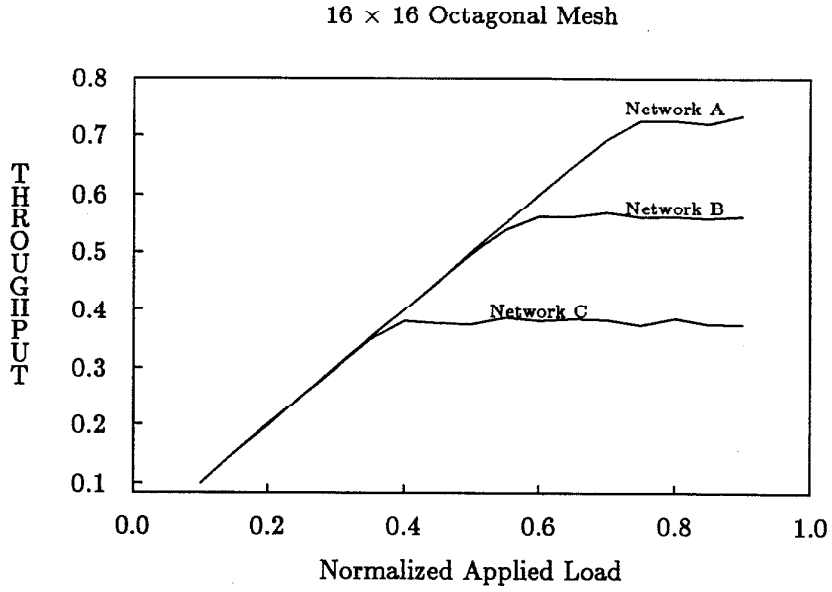

Figure 4.25: Normalized Throughput for Single-packet Message

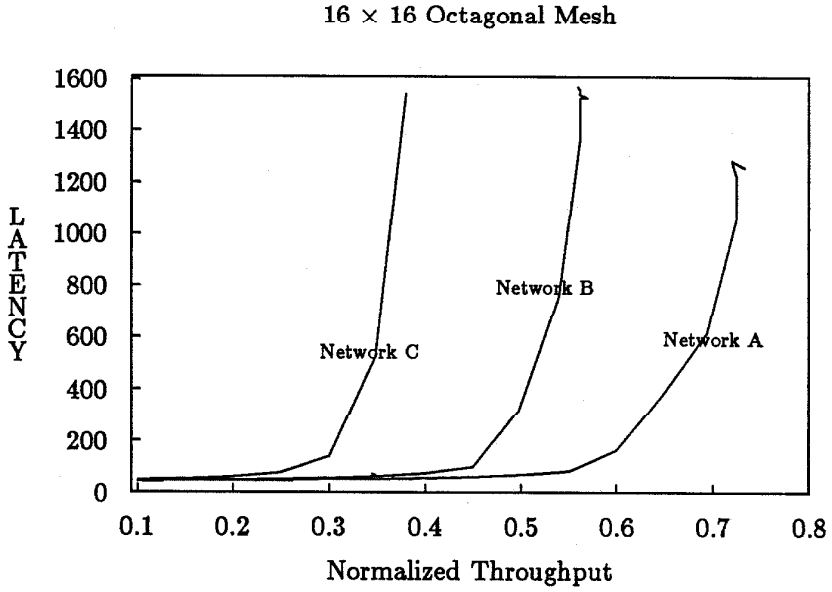

Figure 4.26: Average Latency for Single-packet Message

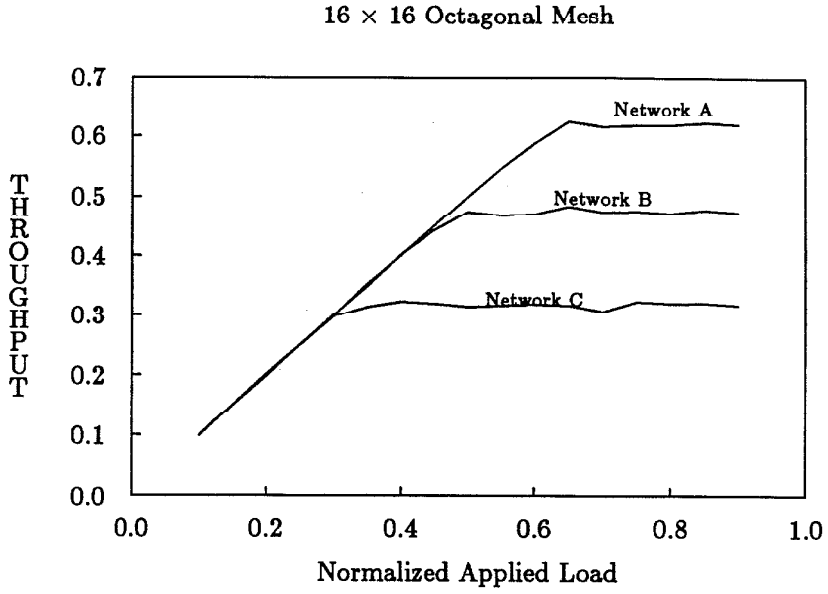

Figure 4.27: Normalized Throughput for Variable-length Message

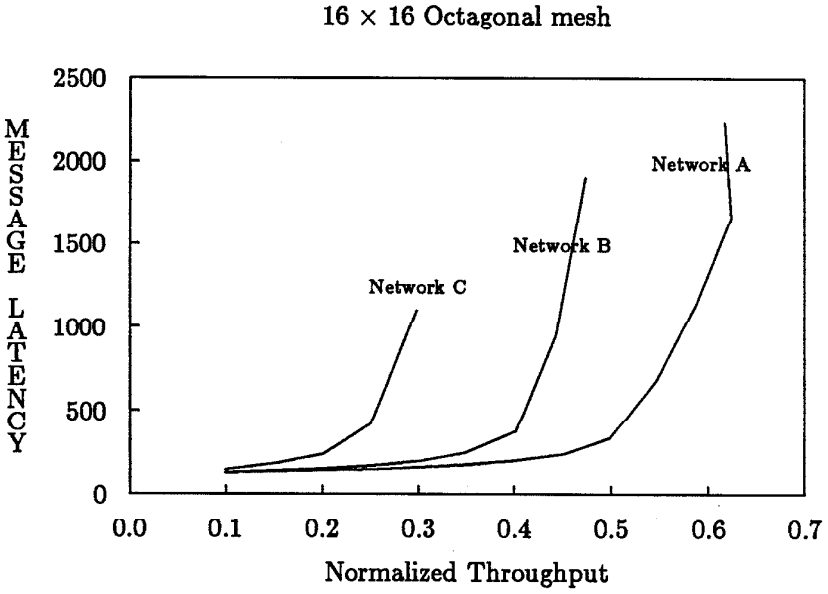

Figure 4.28: Average Latency for Variable-length Message

4.6 Summary

In this chapter, we have examined a number of issues that are fundamental to performing fault-tolerant routing in multicomputer networks. In particular, we have investigated and evaluated the effectiveness of our adaptive routing formulation as a general technique to exploit the inherent path redundancies provided by the richly connected topologies popular in multicomputer networks. Our primary emphasis is on the potential of the adaptive multipath approach rather than in any specific networks.

A simple fault model that captures the salient features of faulty multicomputers was described, and provided the frameworks for our discussion. Motivated by the desire to build high-performance networks through direct hardware realization of the routing operation, we investigated solutions that would allow us to continue to use, with minimal change, the original routing hardware for the non-faulty networks; and, in particular, the notions of convexity, and the communication kernels that characterize the conditions under which we can continue to use the original algorithmic routing relations to systematically direct routing in faulty networks.

The computational complexities involved in determining the fault-tolerant routing configurations suggested by these theoretical notions were studied. In particular, we established the NP-completeness of the problems, and described a simple but effective elimination heuristic procedure to compute the desired fault-tolerant configurations.

The effectiveness of this approach was studied through a set of simulations over the important class of n-dimensional mesh networks. This provided additional insights into the nature of our fault-tolerant routing approach. In particular, based on the simulation results, the 2D octagonally connected mesh network was proposed; this admits excellent fault-tolerant capabilities under the adaptive routing framework. Both the performance and the reliability characteristics of the octagonal mesh were studied in detail. While the simulation results presented in this chapter are still preliminary, and a much more extensive study should be carried out to produce a comprehensive evaluation, the positive results derived from this preliminary study, and the simplicity of this approach, give us hope that the adaptive scheme can be refined to provide a practical basis to support fault-tolerant routing in high-performance multicomputer networks.

Chapter 5

Realization

In the last three chapters, we have examined a variety of theoretical issues that are fundamental to performing adaptive cut-through routing in multicomputer networks. In this chapter, we proceed to examine a number of realization issues concerning a practical implementation for achieving high-speed routing. There are a number of hidden design and implementation problems, as well as some partial solutions in realizing the adaptive-routing approach developed in this thesis. Our objective here is to highlight the various major architectural issues that must be addressed in any practical implementation of the proposed framework, rather than to propose a complete final architectural design. In fact, from the experience in the development and refinement of the oblivious wormhole router, it is clear that many iterations will be required before a highly competitive design of the adaptive router will emerge. Rather, what we are going to describe here summarizes what has been learned, and is intended to serve as a starting point for any serious attempt to implement the adaptive router. This chapter differs from the previous ones in that it raises many more questions than it gives answers; the hope is that these will provoke new ideas.

Specifically, in section 5.1 of this chapter, we shall take a second look at the packet delivery guarantee issue, this time with the emphasis on ***implementation*** rather than on *theoretical* feasibility. We shall argue for a more practical approach based on *congestion control*, a feedback mechanism that is an extension of the injection-fairness-guarantee protocol described in section 2.5. In section 5.2, we shall examine the header-encoding issues in detail, and shall propose possible encoding formats for both the rectilinear and the octagonal meshes; these have a number of desirable properties. This is followed in section 5.3 with a discussion of the buffering issues involved in the storage management of variable-length packets, and a possible solution is presented that can be viewed as a starting point for further investigation. In section 5.4, the various problems involved in performing *fast* adaptive control will be discussed. A possible pipelined control scheme is examined that provides the context for the discussion of the various decision processes. An observation that may lead to certain additional speedup opportunities is presented, with the hope that it will stimulate further thought. In section 5.5, we shall describe a handshake protocol between neighbors that maintains a set of consistent clock signals at each node in a multicomputer network. This allows the use of synchronous logic design techniques, and yet still is arbitrarily extensible. Finally, in section 5.6, we summarize the chapter.

5.1 Congestion Control

The first realization issue we shall discuss concerns the practical assurance of packet delivery. Recall from our feasibility study in Chapter 2 that it is theoretically possible to guarantee delivery of every individual packet in transit inside the network. The scheme calls for the assignment of a priority to each packet, and resolves channel-access conflicts according to the specified ordering. A priority assignment provides information to allow the network to pick *consistent* winners in these conflicts, so that these winners will eventually be delivered. A direct implementation of the suggested priority scheme, however, is rather unlikely as there are several difficult issues yet to overcome. For example, the entire priority field, consisting of the packet's distance from destination and its age, is likely to occupy at least a few flits; hence, a number of cycles will pass before its priority can even be examined, which is somewhat undesirable in fine-grain machines (see section 5.2 for more detail on this). In addition, the requirement to dynamically and consistently increment the *age* parameter is another nontrivial task. However, perhaps the most challenging of all is the priority-discrimination task. With packets dynamically arriving and leaving, resolution of packet priorities will require the maintenance of a dynamically changing *priority queue*, where each comparison is performed on a *multi-flit* data field. Then, on top of all these are the necessary tasks of performing profitable channel assignments, and occasionally misrouting to avoid buffer overflow, both of which are quite complex even without any requirement for following the priority ordering. Clearly, a practical hardware implementation will have to employ something quite a bit simpler.

It is interesting to observe that the congestion-control mechanism employed in the experiments described in section 3.5.4 actually suggests a possible practical alternative that is readily implementable on silicon. In particular, an important property of the congestion-control protocol described there is that it helps to *minimize*, if not completely eliminate, misrouting. As we recall, by allowing misrouting we destroy monotonicity in packet-to-destination distance reduction; this raises the whole issue of guaranteeing delivery. In this sense, the congestion-control protocol described there employs a negative feedback approach to confine the network operating points to regions where misroutings are generally *rare*; hence, the monotonicity in distance reduction is approximately *restored*. The experimental data obtained in those simulations presented strong circumstantial evidence that this negative feedback technique is generally successful.

Another main advantage of employing the congestion-control mechanism is that it also assures approximate fairness in network access. The assurance is no longer absolute because one no longer has absolute assurance of packet delivery, which is fundamental to establishing an absolute fairness guarantee. In other words, the congestion-control protocol is one mechanism that attempts to serve two masters, *ie*, guaranteeing packet delivery and packet injection, but that achieves both objectives only approximately. Whether it is possible to further modify the protocol such that one can restore the absolute guarantee remains an open question.

Implementation of the congestion control protocol is rather straightforward. Following the description in section 2.5.3, each channel needs one extra wire for passing the injection information required to carry out the protocol. Depending on whether or not a node has advanced its injection count, a 1 or a 0 will be passed to its neighbors in

each cycle, according to the protocol specification. Whether the node will be allowed to inject or not in the next cycle depends on the signals received from its neighbors, and on whether misrouting has occurred in this cycle. It is clear that a few simple logic gates are sufficient to generate the required injection-control signals.

In our simulation experiments described in section 3.5.4, channel assignments are performed in a first-come first-served manner; this is a *fair* policy for a dynamically changing set of packets. Sometimes the maintenance of the FCFS ordering may itself be rather expensive. For example, as packets come and go, different buffer locations may be randomly assigned in such a way that a packet's arrival order cannot be directly inferred from the assigned buffer location. In such cases, the simpler round-robin assignment ordering, which is another fair policy, can be used to provide an excellent alternative.

5.2 Header Encoding

The second problem we shall discuss concerns the format of the header of a packet. A main advantage of wormhole routing is its extremely low message latency under relatively light network traffic conditions. This is made possible by having a node forward an incoming packet as soon as enough header information has been accumulated to determine the correct output channel. Performance considerations and the desire for hardware simplicity both suggest that we choose a header format that provides all relevant routing information in the *first* flit of the packet. In an oblivious scheme, where routing proceeds in dimension order so as to avoid deadlock, it is natural to pack the delta values of the corresponding dimensions in that same order. For example, for the rectilinear meshes, we require the following information: ΔX, ΔY for the 2D meshes, and also ΔZ for the 3D meshes. Suppose the oblivious scheme routes a message first along the X dimension, reducing ΔX to zero; then along the Y dimension, reducing ΔY to zero; and finally along the Z dimension, reducing ΔZ to zero; the header will be organized with ΔX in front, followed by ΔY, and then ΔZ. The foremost delta value is decremented by one every time the message header is forwarded one step toward its destination, and when a zero has been detected, the message is switched to the next dimension. Under such a scheme, the message latency, T, under light network traffic conditions, can be expressed as:

$$T \approx \rho D + L,$$

where ρ is the delay per stage required to perform the incremental update, D is the distance the message has to travel, and L is the length of the message. To facilitate the zero-detection and the decrement-by-one operations, a coding scheme that explicitly encodes the notion of a *leading-zero* was developed [14] that allows a message to ripple through an intermediate node with a delay of only two clock ticks, *ie*, $\rho = 2$.

In contrast, for an adaptive scheme that attempts to take advantage of possible alternate routes, it is essential to optimal routing decision to have the direction information of every dimension simultaneously available. In particular, the presence of such direction information in the first flit of the header is extremely desirable in order to allow the assignment logic to make immediate *direct cut-through* routing decisions. One possible scheme is to have the delta value corresponding to each dimension be present explicitly in parallel, starting from the first flit of the header. In addition, we shall need to perform bit-serial increment- and decrement-by-one operations on the encoded

numbers. It is clear that for the adaptive scheme to stay competitive with the oblivious scheme, the delay per stage, ρ, must be kept to a minimum. This is especially true in a fine-grain machine, where the average message distance, D, tends to be large, and the average message length, L, tends to be small. In such cases, ρD will become the dominant component of T; hence, it is essential to keep ρ as small as possible. In the following, we describe an encoding scheme for the adaptive router with a delay per stage which matches that of the oblivious scheme, *ie*, with $\rho = 2$.

5.2.1 Rectilinear Mesh

In this subsection, we shall describe a coding scheme for the rectilinear meshes, assuming the routing relations are defined according to the reduction of a message's L_1-metric from destination. To help simplify the subsequent exposition, we shall first describe the encoding of the delta value for a single dimension. To proceed, we observe that having the *sign* of delta displacement value is necessary and sufficient to determine the profitable direction for that dimension. In particular, for each dimension, an encoding of its sign must be able to distinguish between three possible alternatives: positive-, negative- or zero-delta displacement. This consideration naturally suggests the use of a *sign-and-magnitude* encoding. Furthermore, our need to perform bit-serial arithmetic on the numbers also suggests the use of a *least-significant-bit-first* arrangement. As in the case of the oblivious encoding, let us try to use an alphabet with 3 symbols, $\{0, 1, Z\}$, representing zero, one, and leading-zero. This gives us almost what we want, *ie*, realizing $\rho = 2$, except for the following difficulty: Decrementing a positive magnitude by one; by examining only the first two input symbols; *eg*, it is impossible to distinguish between $+1$, represented as $(\cdots Z Z 1+)$, and $+3$, represented as $(\cdots Z 11+)$. Decrementing the former values gives $(\cdots Z Z Z 0)$, *ie*, zero; while decrementing the latter gives $(\cdots Z 10+)$, *ie*, $+2$. It is clear that the output symbols generated are different in the two cases, and so it is necessary to modify the coding scheme to help distinguish the two cases. One solution is to observe that because the sign is represented explicitly in the very first flit, zero detection can be done readily; thus, there is really no need to encode the leading-zero. Instead, we shall replace the alphabet with $\{0, 1, M\}$, where M explicitly encodes the notion of a *most-significant-one*. Using this new alphabet, $+1$ is represented as $(\cdots 000M+)$, whereas $+3$ is represented as $(\cdots 00M1+)$, with the aforementioned ambiguity removed. Figure 5.1 depicts the sequence of values generated by the decrement-by-one actions. Using this representation, decrement and increment operations both require inspection of two consecutive input symbols in order to determine the next output symbol in the header. This is identical to that required in the oblivious prefix encoding scheme. Figure 5.2 depicts the corresponding update automaton for this representation, where the next output symbols and next automaton states are tabulated, *ie*, described as a *Moore* machine[1]. The symbol I stands for *idle*, and the blank entries represent *don't care* entries that correspond to transitions that *cannot* occur, assuming sufficient flits have been reserved to prevent magnitude overflow. The state transitions for the increment operation are an exact mirror image of the decrement operation, and have been included in the figure for the sake of completeness.

The alphabet described above requires three distinct symbols for the binary repre-

[1] State-output finite-state automaton.

```
NODE  0                                          MOOO+        +8
NODE  1                                        OM11+          +7
NODE  2                                      OM10+            +6
NODE  3                                    OMO1+              +5
NODE  4                                  OMOO+                +4
NODE  5                                OOM1+                  +3
NODE  6                              OOMO+                    +2
NODE  7                            OOOM+                      +1
NODE  8                          OOOOO      <--- TIME         0
NODE  9                        OOOM-                          -1
NODE 10                      OOMO-                            -2
NODE 11                    OOM1-                              -3
NODE 12                  OMOO-                                -4
NODE 13                OMO1-                                  -5
NODE 14              OM10-                                    -6
NODE 15            OM11-                                      -7
NODE 16          MOOO-                                        -8

                    DECREMENT-BY-ONE
```

Figure 5.1: Decrement-by-One Operations Under the {0, 1, M} Alphabet

State	Update Operation, Input Symbol							
	+, 0	+, +	+, -	+, I	-, 0	-, +	-, -	-, I
START	I, PSIGN	I, NSIGN	I, ZSIGN	I, START	I, PSIGN	I, NSIGN	I, ZSIGN	I, START
	+, 0	+, 1	+, M	+, I	-, 0	-, 1	-, M	-, I
ZSIGN	+, POS10				-, NEG10			
PSIGN	+, POS10	+, POS01	+, POS01		+, POS11	+, POS00	0, POS00	
POS00	0, POS00	0, POS10	0, POSMO	0, START	0, POS00	0, POS10	0, POSMO	0, START
POS01	0, POS10	0, POS01	0, POSOM					
POS10	1, POS00	1, POS10	1, POSMO		1, POS00	1, POS10	1, POSMO	
POS11					1, POS11	1, POS00	M, POS00	
POSMO	M, POS00			M, START	M, POS00			M, START
POSOM	0, POSMO							
NSIGN	-, NEG11	-, NEG00	0, NEG00		-, NEG10	-, NEG01	-, NEG01	
NEG00	0, NEG00	0, NEG10	0, NEGMO	0, START	0, NEG00	0, NEG10	0, NEGMO	0, START
NEG01					0, NEG10	0, NEG01	0, NEGOM	
NEG10	1, NEG00	1, NEG10	1, NEGMO		1, NEG00	1, NEG10	1, NEGMO	
NEG11	1, NEG11	1, NEG00	M, NEG00					
NEGMO	M, NEG00			M, START	M, NEG00			M, START
NEGOM					0, NEGMO			

Figure 5.2: Update Automaton for the Sign-and-Magnitude Encoding Using {0, 1, M}

```
NODE  0                                          Z111+      +8
NODE  1                                        Z110+        +7
NODE  2                                      Z101+          +6
NODE  3                                    Z100+            +5
NODE  4                                  ZZ11+              +4
NODE  5                                ZZ10+                +3
NODE  6                              ZZZ1+                  +2
NODE  7                            ZZZZ+                    +1
NODE  8                          ZZZZ0      <--- TIME       0
NODE  9                        ZZZZ-                       -1
NODE 10                      ZZZ1-                         -2
NODE 11                    ZZ10-                           -3
NODE 12                  ZZ11-                             -4
NODE 13                Z100-                               -5
NODE 14              Z101-                                 -6
NODE 15            Z110-                                   -7
NODE 16          Z111-                                     -8

                    DECREMENT-BY-ONE
```

Figure 5.3: Decrement-by-One Operations Under the {0, 1, Z} Alphabet

sentation. A generalization of this scheme, to an arbitrary radix-r representation, will require a total of $2r-1$ symbols: $\{0, 1, M_1, 2, M_2, \cdots, r-1, M_{r-1}\}$, where M_i encodes the most-significant-i symbol. Observe that by using this more general radix-r representation, it is possible to bit-serially subtract any positive value that is $< r$ from the encoded number, while maintaining a delay of two clock ticks per stage. The decrement-by-one operation described in the previous paragraph for the radix-2 representation is just such an example.

On the other hand, if decrement-by-one is the only subtraction operation of interest, as in the present case, it is then possible to devise an alternate solution using the original $\{0, 1, Z\}$ alphabet. This alternate scheme is more economical when generalized to a radix-r representation for $r > 2$. In particular, observe that the distinction between $+1$, $(\cdots ZZZ1+)$, and $+3$, $(\cdots ZZ11+)$, can be made if one is allowed to examine the third symbol as well, since a leading-zero in the representation then will unambiguously identify a $+1$. This motivates the alternate solution: Subtract one from the magnitude representation, so that $+1$ is now represented as $(\cdots ZZZ+)$ and $+3$ is represented as $(\cdots Z10+)$. Because we have subtracted one from the magnitude encoding, we can now unambiguously distinguish $+1$ from larger positive numbers, and -1 from smaller negative numbers; hence, we can update after looking at just the first two symbols. Figure 5.3 depicts the corresponding sequences of values generated by the decrement-by-one operations under this new representation. The advantage of this so-called *sign-first-one-shy-magnitude* representation is that it can be generalized to any arbitrary radix r using an alphabet of size $r+1$ instead of $2r-1$: $0, 1, \cdots, r-1$, and Z. This could lead to substantial savings when representing a large number.

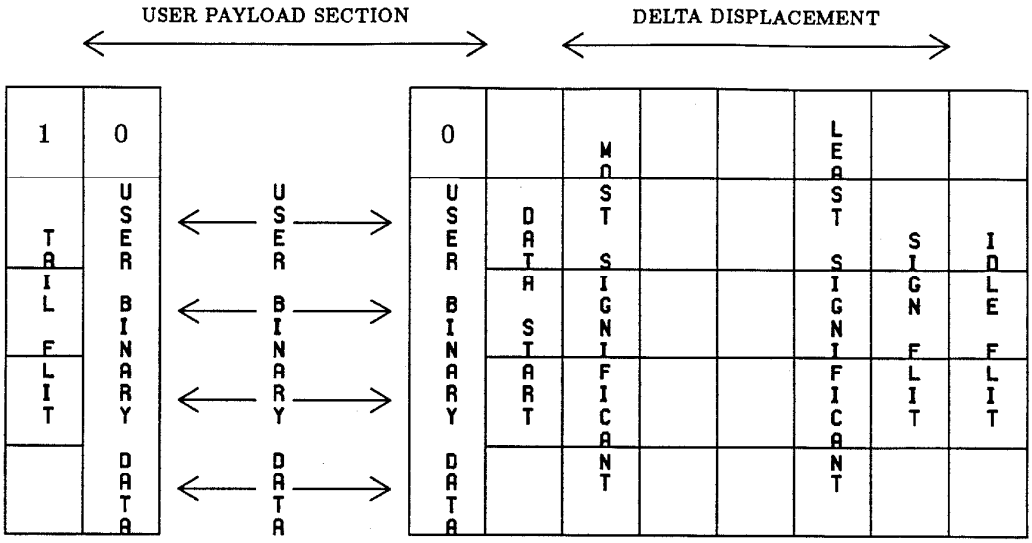

Figure 5.4: A Five-Bits Packet-Encoding Layout for a 3D Rectilinear Mesh

In addition to the encoding for the delta values, we would also need at least three other encodings: I for the *idle* flits, D to mark the start of the user *data* section, and T to represent the tail flit. Since each dimension requires its own delta displacement value, and since each dimension induces at least three distinct alternatives (0, 1, and, Z or M), a format for a 3D mesh would require a total of $3^3+3 = 30$ different symbols. These can be nicely represented using five bits. Figure 5.4 depicts a possible arrangement using such an encoding for the 3D mesh. These, together with the additional 1 bit required to carry out the congestion-control protocol, take a total of six bits per channel. Similarly, one can use the same five-bit wide format for a 2D mesh that explicitly encodes ΔX, ΔY, and $|\Delta X| - |\Delta Y|$, all of which change by ± 1 across channels. The sign of the last quantity, *ie*, the difference in magnitude, enables efficient discrimination of the maximum of $|\Delta X|$ and $|\Delta Y|$. Having this information allows one to implement the optimizing heuristic mentioned in Chapter 3.

5.2.2 Octagonal Mesh

In this subsection, we extend our previous discussion to consider the encoding problem for the 2D octagonal mesh, which has been demonstrated in Chapter 4 to display some nice fault-tolerant potential. In particular, we shall assume the routing relations to be defined according to the reduction of the combined metric that is the summation of the L_1-metric and the L_∞-metric. In other words, the set of profitable channels at any intermediate node is the ***union*** of those that reduce the L_1-metric with those that reduce the L_∞-metric.

As was mentioned in the previous subsection, it is extremely desirable to have explicit direction information for all the profitable channels simultaneously available in the first flit of the packet header. There are several different ways in which to organize the header so as to provide the necessary information. In particular, it is interesting to observe that:

- The profitable channels induced by the reduction of the L_∞ metric, when restricted to the non-diagonal channels, are a *subset* of those induced by a reduction of the L_1-metric.

- For the diagonal channels, the reverse is true; *ie*, the profitable channels induced by the reduction of the L_1 metric, when restricted to the diagonal channels, are a subset of those induced by a reduction of the L_∞-metric.

For example, the first observation indicates that having the ΔX and ΔY displacement values is *sufficient* to cover the required direction information for the horizontal and the vertical channels. The second observation indicates that it is possible to *decouple* the diagonal and the non-diagonal channels from each other, and to encode the required direction information for the two sets separately.

To obtain the direction information for the four diagonal channels, it is clear that we must maintain extra information. For our present purpose, two sets of quantities that naturally provide the required information are the $\Delta(X+Y)$ and $\Delta(X-Y)$ displacement values, and the magnitude difference $|\Delta X|-|\Delta Y|$. In particular, Figures 5.5 and 5.6 illustrate how these extra delta quantities can be used to identify the profitable diagonal channels. The delta-sum and delta-difference encodings give explicit direction information for the diagonal channels, independent of the ΔX and ΔY encodings. On the other hand, the delta-magnitude-difference encoding requires some extra decoding in order to extract the desired direction information. For our purpose, we shall choose the latter because it has one fewer value to maintain. Observe that the *sign* of these quantities gives us almost what we want. However, by looking at only the first symbol, one cannot distinguish $+1$ from larger positive numbers, or -1 from smaller negative numbers. In particular, having the delta-sum or delta-difference equals to ± 1 implies that both channels of the corresponding diagonal axis are *nonprofitable*. For example, with the delta-sum equal to $+1$, sending the packet across the south-west channel increases d_{L_∞} by 1, and d_{L_1} by 2, whereas sending it across the north-east channel leaves both metrics unchanged. In other words, they are indistinguishable from zero for the purpose of direction discrimination.

We now proceed to describe the possible encodings for these extra delta quantities so that the required direction information is given correctly and explicitly in the first flit. First of all, from the above definitions, it is clear that one may be required to perform bit-serial increment-by-two and decrement-by-two operations on these quantities when the packet is forwarded across a diagonal channel. As we have seen in the previous subsection, this can be accomplished with $\rho = 2$, if we use an alphabet $\{0, 1, M_1, 2, M_2, \cdots, r-1, M_{r-1}\}$ with $r \geq 3$. To help distinguish $+1$ from zero, an immediately obvious solution is to invent two new special symbols: P for $+1$, and N for -1. In particular, let us assume that we shall use a radix-4 representation, using $\{0, 1, A(\equiv M_1), 2, B(\equiv M_2), 3, C(= M_3)\}$. Figure 5.7 illustrates the decrementing sequences. Hence, using this radix-4 encoding for $|\Delta X|-|\Delta Y|$, together with the encoding of ΔX and ΔY, the required directional discrimination for the diagonal channels can be carried out by examining only the first flit of the header. In particular, since the radix-4 encoding for the ΔX or ΔY value requires only five symbols, $\{0, 1, 2, 3, Z\}$, using one-shy magnitude encoding, the entire triplet $(\Delta X, \Delta Y, |\Delta X| - |\Delta Y|)$ requires $5 \times 5 \times 7 = 175$ symbols. This can fit nicely into a byte-wide flit width. It is also inter-

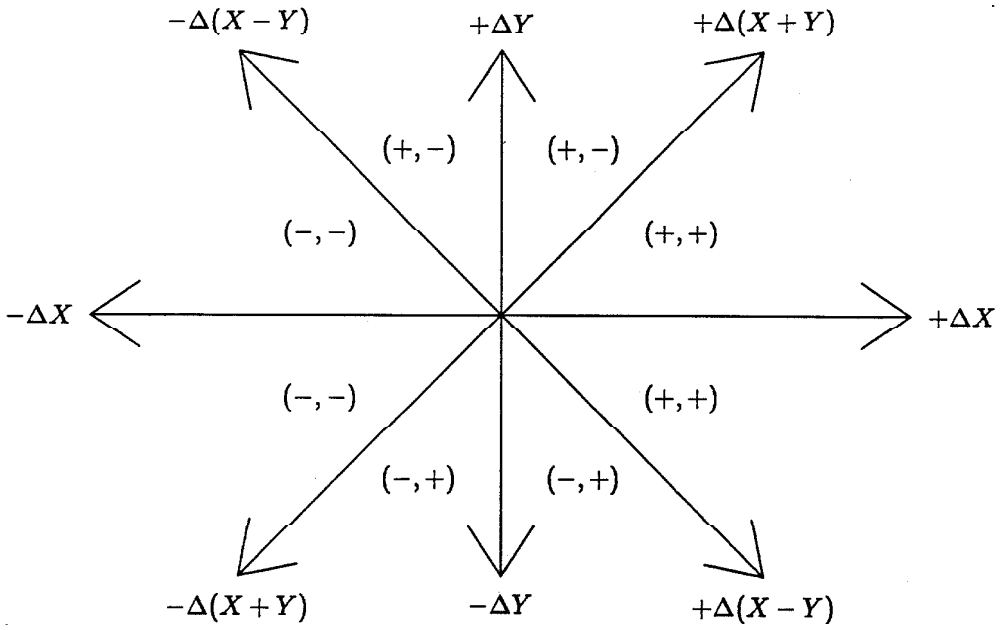

Figure 5.5: Diagonal Channel Encoding: Signs of $\Delta(X+Y)$ and $\Delta(X-Y)$

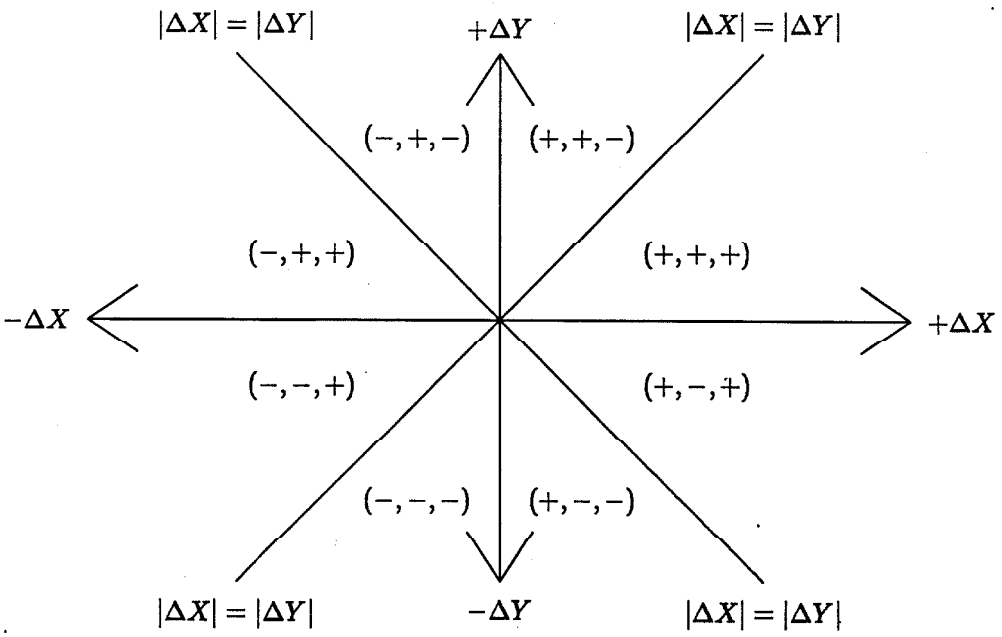

Figure 5.6: Diagonal Channel Encoding: Signs of $\Delta X, \Delta Y$, and $|\Delta X| - |\Delta Y|$

```
NODE  0                                        00B0+        +8
NODE  1                                      00A3+          +7
NODE  2                                    00A2+            +6
NODE  3                                  00A1+              +5
NODE  4                                00A0+                +4
NODE  5                              000C+                  +3
NODE  6                            000B+                    +2
NODE  7                          000AP                      +1
NODE  8                        00000      <--- TIME         0
NODE  9                      000AN                          -1
NODE 10                    000B-                            -2
NODE 11                  000C-                              -3
NODE 12                00A0-                                -4
NODE 13              00A1-                                  -5
NODE 14            00A2-                                    -6
NODE 15          00A3-                                      -7
NODE 16        00B0-                                        -8
                 DECREMENT BY ONE

NODE  0                                  00B0+              +8
NODE  1                                00A2+                +6
NODE  2                              00A0+                  +4
NODE  3                            000B+                    +2
NODE  4                          ZZZZ0      <--- TIME       0
NODE  5                        000B-                        -2
NODE  6                      00A0-                          -4
NODE  7                    00A2-                            -6
NODE  8                  00B0-                              -8
              DECREMENT BY TWO (EVEN SEQUENCE)

NODE  0                                  00B1+              +9
NODE  1                                00A3+                +7
NODE  2                              00A1+                  +5
NODE  3                            000C+                    +3
NODE  4                          000AP      <--- TIME       +1
NODE  5                        000AN                        -1
NODE  6                      000C-                          -3
NODE  7                    00A1-                            -5
NODE  8                  00A3-                              -7
NODE  9                00B1-                                -9
              DECREMENT BY TWO (ODD SEQUENCE)
```

Figure 5.7: Decrementing the Radix-4 Representations

esting that there are many curious combinations of the different alphabets mentioned here. For example, it is possible to perform decrement-by-two operations with $\rho = 2$ using the alphabet $\{0, 1, Z, M\}$ with one-short-magnitude encoding. The final choice will thus depend on the automaton-implementation complexity.

In summary, by encoding ΔX, ΔY, and $|\Delta X|-|\Delta Y|$ in parallel, all direction information necessary for guiding routing in the octagonal mesh network under the composite $L_1 + L_\infty$ metric is available in the very first flit of the header. Furthermore, all necessary updates can be performed with a delay of two clock ticks per stage, giving $\rho = 2$.

5.3 Storage Management

It is clear from our study of the stochastic model in Chapter 3 that the extra performance gain achieved in changing from the oblivious wormhole technique to adaptive cut-through switching requires the investment in more silicon area. The adaptive technique offers the opportunity and ability to trade silicon area for channel utilization more efficiently. The largest component of circuitry is the internal storage. Internal buffers are required for storing packets temporarily while they are waiting for their respective profitable channels. These waiting packets form the pool of candidates that compete for output channel assignments. Since the matching statistics are better for larger pool sizes, it is desirable to have many buffers. In practice, the number that one can afford depends on the amount of available silicon area and the complexity of the buffer circuitry.

5.3.1 Bounded-Length Message Packets

In all our discussions so far, we have adopted the fixed-packet-length assumption, which simplifies the foregoing exposition of the fundamental ideas behind the described framework. It is, however, not necessarily convenient, since messages sent by different processes or objects generally do not contain the same amount of data, nor do they always come in with lengths that are exact multiples of the chosen fixed-packet length. As a result, many packets must be padded with null characters to the next rounded-up multiple of L, thereby wasting network bandwidth. A better alternative is to replace the fixed-packet-length assumption with the *bounded*-packet-length assumption. Under this assumption, packets may have variable lengths, subject to the restriction that packet lengths be $\leq L$. Long messages are partitioned into multiple packets of size up to L; short messages are simply transmitted as single packets of length $l \leq L$. This eliminates internal fragmentation, and allows much more efficient use of available network bandwidth. As usual, the maximum packet length, L, is determined as a tradeoff between apparently contradictory objectives:

- Short maximum length — to more efficiently exploit the multiple paths allowed in adaptive routing if possible, and, in particular, to reduce the amount of buffering resources required at each node.
- Long maximum length — to reduce the inefficiencies caused by the overhead necessary to carry the routing header information. The smaller average number of packets per message also reduces the pressure on message reassembly at the destination nodes.

For example, consider the message traffic distribution used in our performance simulation experiments in Chapter 3. Setting $L = 128$ flits instead of 32 flits reduces the fraction of fragmented messages from 99.6% to $\approx$ 15%; whereas setting $L = 256$ flits further reduces the fraction to less than 0.1%, nearly eliminating message reassembly overhead. In the following subsection, we shall highlight the problems involved, and describe a possible storage structure for handling such variable but bounded-length packets that could be viewed as a starting point for further investigation.

5.3.2 Bounded-Length Packet Storage

Let us take a closer look at some of the problems that arise when we allow variable-length packets in our network. Since packets can have length $\leq L$, many more variable-length packets may require storage than fixed-length packets. This can occur because the output channels may be transmitting previously stored long packets while short packets are continuously arriving at the input channels during the same period. For example, consider a 16×16 2D mesh with a typical L of 256 flits, and a minimum packet length of 6 flits (1 sign flit, 1 tail flit, and 4 delta displacement flits for a 16×16 mesh). In the worst case, a router may be required to temporarily store hundreds of *short* packets. It is clear that simply keeping track, let alone managing the storage, of such a large number of packets simultaneously is already a formidable task, as is performing assignment optimization on these many packets. Clearly any acceptable storage structure would have to keep down the complexity of the corresponding control logic.

In order to keep the assignment logic reasonably simple, we shall restrict outside access to the storage structure to a small fixed number of ports. In particular, we shall assume the following:

1. Storage is organized as a collection of b FIFO buffers of equal size. FIFO buffers are readily implementable in VLSI.

2. Each FIFO buffer has exactly one input and one output port; this permits simultaneous reading and writing.

3. A FIFO buffer can be used to simultaneously store a multiple number of packets.

Figure 5.8 in the next section depicts a possible conceptual layout of the adaptive router, where the internal buffer pool is organized as a collection of FIFI buffers. Using FIFO buffers, the assignment logic need not explicitly keep track of the stored packets, and there will always be a small fixed number of candidate packets at the output ports that may be considered for optimization. In particular, every stored packet will eventually emerge at one of the output ports, at which time it will become eligible for output-channel assignment. Similarly, the input control logic need not explicitly keep track of where the empty spaces are. Rather, there will always be a small fixed number of input ports where incoming packets may be allocated for storage. Furthermore, it helps to minimize the coupling between the buffer inputs and the buffer outputs, allowing the input and output control logic to operate almost independent of each other. However, cooperation must still exist between the two controllers to prevent buffer overflow when the buffer is filled to capacity.

For a *limited-access* storage structure like the set of FIFO buffers assumed here to function properly, it has to satisfy the following requirements:

(a) Each packet should always have a chance to wait for its own profitable channels when it emerges at the output port.

(b) Whenever the storage is filled to capacity, valid data must be present at a sufficient number of output ports so that misrouting can proceed as necessary to prevent buffer overflow.

To satisfy these requirements, we shall use a set of FIFO buffers of total capacity B' to *emulate* a buffer structure of a smaller capacity B with the desired properties. In essence, this scheme allows us to trade for a more efficient communication bandwidth usage by a somewhat inefficient use of memory.

We now describe the storage management policy for these FIFO buffers. Whenever the input control wants to allocate a buffer to a newly arrived packet, it is necessary to locate a FIFO with an idle input port. Furthermore, the selected FIFO must either be empty or have sufficient empty space to store a maximum-size arriving packet. This is necessary in order to decouple the input allocation logic from the output assignment logic. This decoupling is necessary because we allow each buffer to simultaneously store a multiple number of packets. Since the output assignment logic has *no* access to the stored packets before they reach their respective output ports, in order to allow each a chance to wait for its profitable channel, it must be possible for each packet to wait for their profitable channels if so determined by the assignment logic. This in turn requires the buffer to have sufficient empty space to store the newly allocated packet, which may be of maximum size.

For the sake of description, let us assume that the FIFO buffer length is equal to kL, where $k > 1$ is some small positive number. Let B denote the capacity of the simulated storage structure. Observe that since the variable length packets may terminate any time during input, the storage structure must guarantee the existence of at least d buffers with $\geq L$ empty space to store new packets at *all* times. In general, with no *a priori* knowledge of the output control decisions, a maximum of B flits of stored packet data can be scattered arbitrarily among the b FIFO buffers. We now observe that as long as $B < (b-d+1)((k-1)L+1)$, we can be sure that there is always at least d buffers with sufficient room to hold a maximum size packet. This establishes an upper bound on B, given the total number and the length of the FIFO buffers used in order to satisfy requirement (a). On the other hand, the preemption requirement, *ie*, requirement (b), dictates that when the storage structure is filled to its full capacity B, there are at least $d+1$ nonempty buffers, so that the privileged packet can always be retained for transmission later. Again, the necessity to accommodate any arbitrary data distribution among the b buffers implies that we must have $B > dkL$. To summarize, we have derived the following *structural* requirements:

$$d\,k\,L < B < (b-d+1)((k-1)L+1).$$

The number of FIFO buffers, b, should be chosen to be small enough to admit practical implementation, and large enough to deliver acceptable performance. From the simulation results obtained in Chapter 3, having $b \geq 10$ appears to deliver reasonable performance. To get a better feeling of the typical amount of storage required under this scheme, let us assume for the 2D mesh network that $b = 16, k = 3, L = 128$ flits, and $d = 5, w = 5$ bits. With these parameters, we can pick $B = 1920$ bytes. In other words,

this scheme employs a total of 3840 bytes of storage organized into sixteen FIFO buffers of size 240 bytes each to simulate the desired limited-access storage structure of capacity of 1920 bytes. This gives a simulation efficiency of exactly 50%, and represents the typical overhead paid to accommodate the irregularity introduced by variable-length packets under this scheme. Higher efficiencies can be obtained if the number or the length of the buffers is increased. For medium-grain machines such as the Symult Series 2010 or Intel iPSC/2, where each node typically has several megabytes of memory, this represents an insignificant factor of $\approx 0.1\%$ in silicon real estate investment to improve network performance. For fine-grain machines, however, the amount of storage required may itself constitute a large fraction of the total silicon area per node. In such case, one may have to go back to fixed-length packets in order to conserve silicon area. Whether better solutions that require less storage, while simultaneously satisfying both requirements (a) and (b), will be found remains an open question.

5.4 Adaptive Control

Given that all of the routing direction information is present at the very first flit of the packet header, and given a suitable set of FIFO buffers, how to perform the desired adaptive assignments becomes the next natural design issue to consider. At issue is how to perform the packet-to-channel routing assignment *fast* enough that it does not itself become a new bottleneck in the system. In addition, there is also the need to allocate buffers to the arriving packets that require temporary storage. In this section, we shall describe a tentative control scheme for performing these tasks. Our main purpose is to use the proposed scheme to provide a context to highlight and discuss the various problems involved.

5.4.1 A Pipelined Control Scheme

Since it is clear that the adaptive control process is rather complicated, a *pipelined* approach is called for in order to keep the cycle time acceptable. Conceptually, the adaptive control process can be nicely partitioned into a number of sequential phases:

1. *Buffer Allocations*: Upon receipt of each newly arrived packet, the first step is to allocate an available internal buffer for its temporary storage. It is possible to avoid certain unnecessary allocations, *eg*, in the case of a direct cut-through, by deferring this stage until after the assignment has been computed. However, performing the allocation first helps to streamline the data flow, which can simplify the layout and the control logic considerably.

2. *Profitable Assignments*: The second phase attempts to aggressively assign as many profitable channels to the competing packets as possible by having each packet bid for its own profitable output channels. Here the competitors include those that have already been buffered previously and those that have just arrived.

3. *Misrouting Assignments*: Given the profitable assignment results obtained from the previous step, the current step then attempts to fill in the remaining assignments by defensively misrouting packets in cases where that is required in order

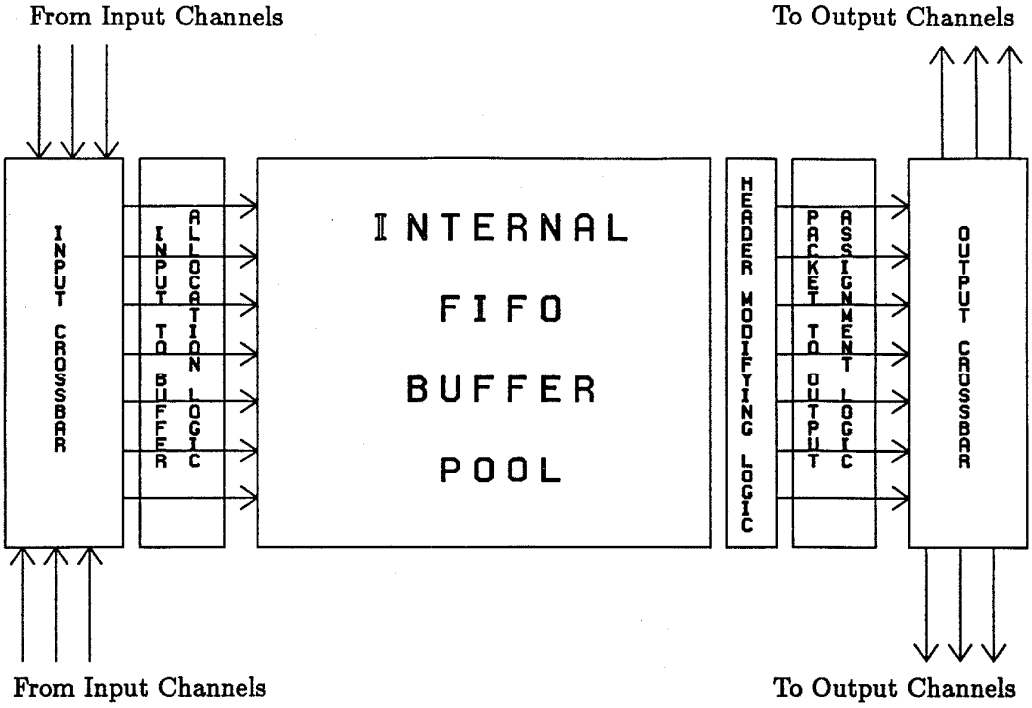

Figure 5.8: A Possible Conceptual Layout of the Adaptive Router

to prevent buffer overflow. The result then becomes the final packet-to-channel assignments according to which the buffered packets will be sent out.

4. *Header Modifications*: This stage updates the encoded header according to the direction it is being sent out; the result from this stage will then be driven across the chip boundary into the next router. This modification has to occur last because a header can be correctly modified, *ie*, incremented or decremented or not changed, only *after* the correct assignment has been computed.

This four-stage partitioning of the adaptive control process appears to make it a natural candidate for pipelining. Figure 5.8 shows a possible conceptual layout of the datapath and control structures for the adaptive router. At the input side, incoming packets arriving from the input channels are switched into the FIFO buffers through the input crossbar, where they are temporarily stored, and wait for their turn to be switched out. At the output side, packet-to-channel assignments are computed for all the buffered packets, and the assigned packets are then switched out to the output channels through the output crossbar. Although the input and output control logic are drawn in between the two crossbars and the internal buffers, in practice, they are likely to be physically meshed into the correspondingly controlled crossbars.

Figure 5.9 depicts a possible conceptual structure of the control pipeline for the adaptive router, where the four stages correspond exactly to the four control tasks specified above. Assuming that one employs the canonical *two-phase nonoverlapping* clocks in a silicon implementation, the four stages can be computed in two clock cycles:

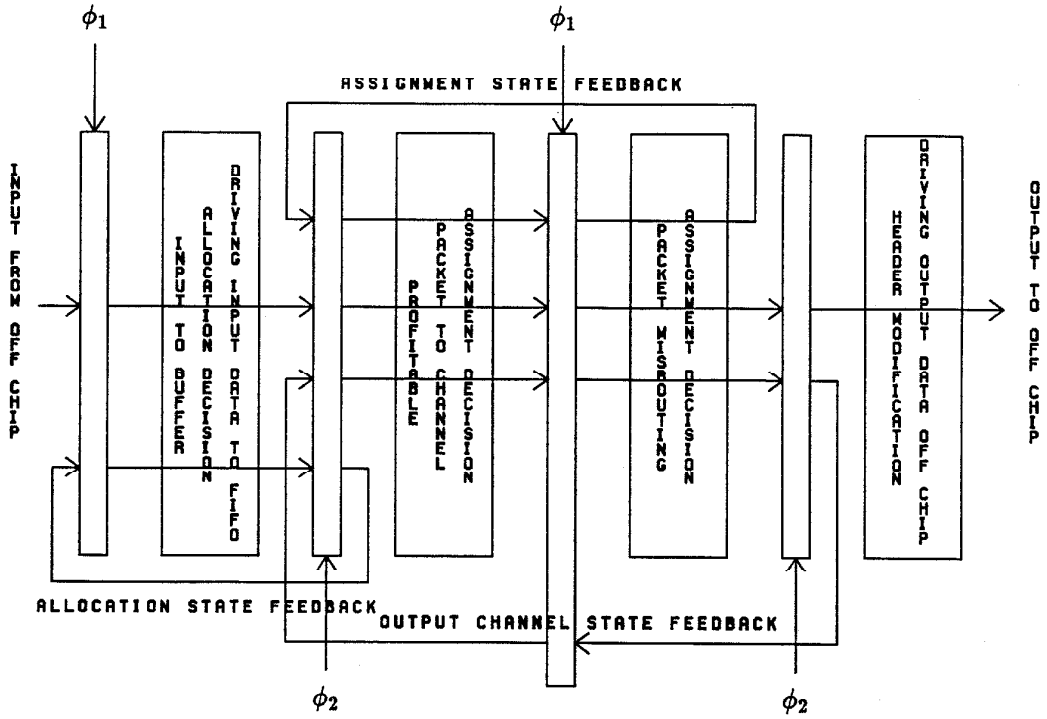

Figure 5.9: A Conceptual Pipelined Control Scheme for the Adaptive Router

buffer allocations during ϕ_1, and profitable assignments during ϕ_2 of the first cycle; misrouting assignments during ϕ_1, and header modifications and driving signals off-chip during ϕ_2 of the second cycle. It is important to observe that since the header modification process already induces a necessary delay of two clock cycles, propagating the control flow through this pipeline will not incur extra delay, and therefore will not slow down the packet data flow. In particular, packet data will be latched into the FIFO storage during ϕ_1 of each clock cycle. Thus, by ϕ_2 of the second cycle, each candidate packet will have at least two flits around to feed the modification circuit, which should now be implemented as a *Mealy* machine[2].

Figure 5.9 shows a number of internal-state feedback paths within the control pipeline. For example, the assignment states are updated alternatively by both the profitable assignment logic and by the misrouting assignment logic, and the output of one serves as input for the other. Likewise, changes in output channel state are triggered by tail-flit detection circuits during ϕ_1 of a cycle, and are fed back to the assignment decision logic during ϕ_2 of the second cycle. This allows other packets to be assigned and driven off chip during ϕ_2 of the third cycle, assuring continuous nonstop operation of the pipeline. In fact, all the necessary information needed by stages that are active during ϕ_2 is computed by stages that either are active during ϕ_1 of the same cycle or are active during ϕ_2 of previous cycle, and *vice versa*.

It is interesting to observe that the complexities of these different pipeline stages can be made relatively well balanced. In the first stage, the buffer allocations are computed

[2]Transition-output finite-state automaton.

and the input signals are driven across the input crossbar into the FIFOs; both will require a nontrivial amount of time. In the final stage the headers are modified and the signals are driven across the output crossbar, and then across the chip boundary. Header modifications can be done quickly; if driving signals off chip takes approximately the same amount of time as computing the buffer allocations, the first and the last stage will be balanced. Similarly, computing the profitable assignments and the misrouting assignments (which also keeps track of the misrouting quota) are likely to take approximately equal amounts of time. On the other hand, both assignments are likely to be more complicated than just buffer allocation. Hence, if the delay of the assignment logic can be kept to at most *twice* that of the buffer allocation logic, all four stages will be well balanced.

5.4.2 The Decision Logic Structures

We observe that because the buffer allocation and the channel assignment decisions can be considered to be mirror images of each other in that they are conceptually very similar, they may be grouped together and discussed under a generic decision-process framework. It is rather clear that the most complicated decision process is that of computing the profitable packets-to-channels assignments. The computation of these profitable assignments may be regarded as the true *productive* work done by the adaptive control, whereas the others are just housekeeping overhead. Hence, we shall discuss the profitable channel assignments decision in detail; this will serve as our prime example of the generic decision process.

Mathematically, an *optimal* solution to this assignment task is identical to the *maximum matching* problem of a bipartite graph, where the two vertex sets are the unassigned packets and the available channels; and the edge set is the set of profitable channels for each packet. While the maximum matching problem admits rather simple sequential algorithmic solutions [43], it is clear that a direct hardware implementation of such exact solutions remains unrealistic. An excellent approximating alternative that is algorithmically much simpler is to find instead a *maximal* matching. Figures 5.10 to 5.13 compare the matching statistics for the 2D and 3D torus obtained under the two matching schemes, where the profitable-channel distributions are assumed to be isotropic as in section 3.3.1. A maximal matching can be computed using a very simple one-pass sequential assignment process in which the scan can proceed in the desired round-robin fashion. In particular, a sequential scan over the set of competing packets provides a very convenient context for each packet to make further discriminatory selection among its own set of profitable channels, *eg*, as required in the implementation of the performance-enhancing heuristic mentioned in section 3.2.

The desired maximal assignment decisions can be implemented very naturally by using an iterative decision network. At every cycle, the partially formed assignment decision will ripple around the chain, picking up any profitable assignments along its way, until it gets back to its starting position, at which point, the decision has been completed. If further discrimination among the profitable channels is desired, it can be implemented using a *bilateral* iterative network [21]. Figure 5.14 depicts such a possible structure, where the shaded cells represent the starting scanning positions. The assignment state inputs are gated into the network during ϕ_2, and the updated assignment state outputs will be latched out during ϕ_1. In any case, the starting positions will be cyclically shifted

2D TORUS	*Number of Packets*									
Matching Size	1	2	3	4	5	6	7	8	9	10
1	1.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
2	0.0000	1.0000	0.0624	0.0156	0.0036	0.0009	0.0002	0.0001	0.0000	0.0000
3	0.0000	0.0000	0.9376	0.2835	0.1369	0.0664	0.0318	0.0154	0.0081	0.0039
4	0.0000	0.0000	0.0000	0.7009	0.8595	0.9327	0.9680	0.9845	0.9919	0.9961

Figure 5.10: Maximum Matching Probabilities for a 2D Network

2D TORUS	*Number of Packets*									
Matching Size	1	2	3	4	5	6	7	8	9	10
1	1.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
2	0.0000	1.0000	0.1877	0.0470	0.0116	0.0029	0.0007	0.0002	0.0000	0.0000
3	0.0000	0.0000	0.8123	0.5466	0.3083	0.1626	0.0838	0.0428	0.0214	0.0105
4	0.0000	0.0000	0.0000	0.4064	0.6801	0.8345	0.9155	0.9570	0.9786	0.9895

Figure 5.11: Maximal Matching Probabilities for a 2D Network

3D TORUS	*Number of Packets*									
Matching Size	1	2	3	4	5	6	7	8	9	10
1	1.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
2	0.0000	1.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
3	0.0000	0.0000	1.0000	0.0011	0.0003	0.0000	0.0000	0.0000	0.0000	0.0000
4	0.0000	0.0000	0.0000	0.9989	0.0154	0.0034	0.0008	0.0002	0.0001	0.0000
5	0.0000	0.0000	0.0000	0.0000	0.9843	0.1097	0.0515	0.0237	0.0124	0.0058
6	0.0000	0.0000	0.0000	0.0000	0.0000	0.8869	0.9477	0.9761	0.9875	0.9942

Figure 5.12: Maximum Matching Probabilities for a 3D Network

3D TORUS	*Number of Packets*									
Matching Size	1	2	3	4	5	6	7	8	9	10
1	1.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
2	0.0000	1.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
3	0.0000	0.0000	1.0000	0.0696	0.0087	0.0011	0.0002	0.0000	0.0000	0.0000
4	0.0000	0.0000	0.0000	0.9304	0.2678	0.0740	0.0196	0.0051	0.0013	0.0003
5	0.0000	0.0000	0.0000	0.0000	0.7235	0.5623	0.3367	0.1838	0.0952	0.0488
6	0.0000	0.0000	0.0000	0.0000	0.0000	0.3626	0.6435	0.8111	0.9035	0.9509

Figure 5.13: Maximal Matching Probabilities for a 3D Network

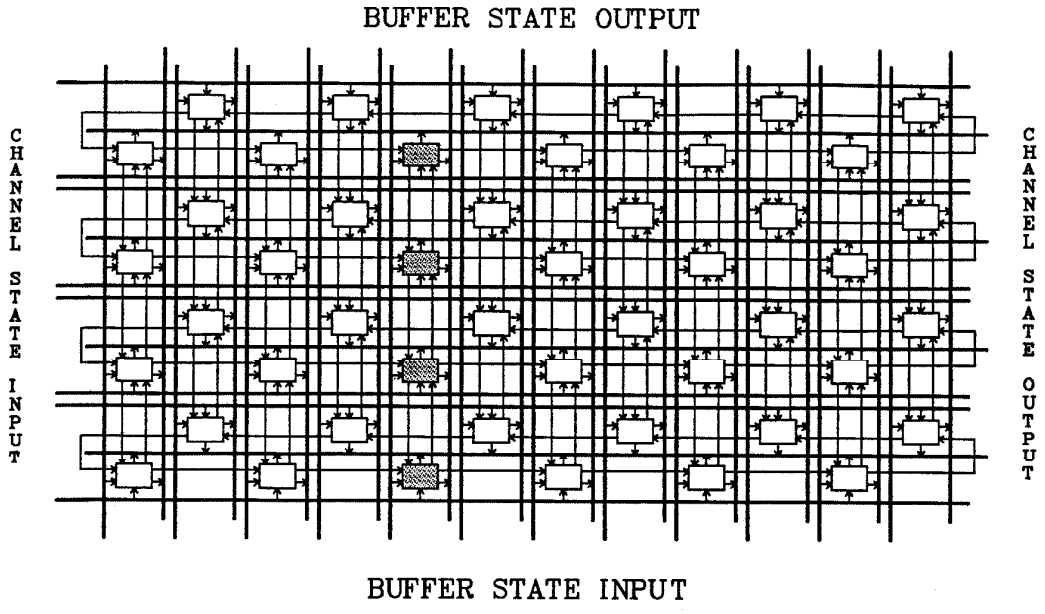

Figure 5.14: A Bilateral Iterative Decision-Network for Profitable Assignments

by one position for the next clock cycle. This implements the desired round-robin scan, which, in the absence of misrouting, assures fairness in making profitable assignments.

The decision logic for computing the misrouting assignments, and that for allocating internal buffers both can be implemented using similar unilateral iterative logic structures. For the misrouting assignments, a misrouting quota can ripple along with the partial decision. The quota will decrement every time a successful misrouting is committed; when it reaches zero, further misrouting is disabled. Decrementing can be accomplished easily with a *shift*, if the quota is generated in a unary representation. Again, these scans should proceed in a round-robin fashion in order to generate more balanced buffer allocation and misrouting assignment distributions.

The main advantage of such iterative decision-network implementation is that it is very well matched to the topological layout of the underlying datapath; this is very desirable in a VLSI implementation. In particular, it is possible and perhaps advantageous to embed the iterative logic structure inside the crossbar, which is likely to save some silicon area. The main disadvantage of such an iterative structure is that it could be rather slow. For example, during each active clock phase, the partial decisions have to ripple through all the buffer entries. As a result, the rippling delay will be proportional to the total number of internal buffers. Hence, the challenge here is to design the logic gates *fast* enough so that the total rippling delay will not become excessive, and therefore slow down the clock. Otherwise, a different faster decision structure will have to be found.

5.4.3 A Speedup Opportunity

An additional speedup opportunity exists which, if properly exploited, may shorten the average cycle time. Following the discussion in the performance chapter, we realize that statistically, during a majority of the routing cycles, the adaptive router is just *passively* switching flits without making any new decisions. Furthermore, during those *active* cycles when decisions have to be made, they are most likely to be associated with single-packet arrivals or departures.

To get a better feel for the likelihood of having multiple headers arrive at the same cycle, let us derive a rough approximation of the desired probability: Assume that the average packet length is l flits. Then the probability, P_m, of having two or more packet headers arrive during the same cycle under an average channel-utilization factor p, assuming independent memoryless, (*ie*, *Bernoulli*) arrival processes, will be:

$$\begin{aligned} P_m &= 1 - P_0 - P_1 \\ &\approx 1 - \left(1 - \frac{p}{l}\right)^c - c\left(\frac{p}{l}\right)\left(1 - \frac{p}{l}\right)^{c-1} \end{aligned}$$

For example, for $c = 5, l = 32$, and $p = 0.7$, *ie*, at the performance transition point, we have $p_m \approx 0.0046$. In other words, the chance of having multiple header arrivals is approximately 0.5%, even under extremely heavy traffic! On the other hand, there is $\approx$ 10% chance that there will be exactly one new header at each cycle, and that the remaining $\approx$ 90% of the cycles will have zero arrival. In fact, the longer the average packet length, or the lower the channel-utilization factor, the less likely is the occurrence of arrival or departure.

One immediate consequence is that these statistical variations will lead to an average cycle time that is lower than the worst-case cycle time. To get a rough idea of the kind of speedup possible, let us consider a simple *two-tier* model, where the cycle time falls into only two hypothetical figures, either 20ns or 100ns, and nothing else. A 16×16 2D network that follows the coherent-channel protocol, where the cycle times of its individual elements, which are picked probabilistically between the two delay values, are simulated. Figure 5.15 plots the average cycle delay time for a network that follows the coherent-channel protocol, versus the percentage of active cycles, *ie*, those having long cycle delay. Also shown in the figure is the curve that plots the naively computed average, *ie*, those values computed by ignoring any interaction between adjacent nodes, which serve as an obvious lower bound for comparison with the simulated values. Since the percentages will be less than 10% for all reasonable packet lengths and feasible network throughputs, these simulation results show that an implementation that can exploit these statistical variations may indeed make it possible to gain $\approx$ a factor of 2 in speed.

If one is to implement the decision processes in asynchronous handshake logic [33,48], then it is possible to exploit the statistical variations in a rather straightforward manner. By generating explicit completion signals in each of the decision processes, the decision logic will return almost immediately in a majority of the cycles, *ie*, the passive cycles. Even for cycles with single-packet arrivals or departures, the decisions can often return much faster than the worst-case delay, which must be accounted for in a synchronous implementation. On the other hand, a synchronous implementation is likely to consume

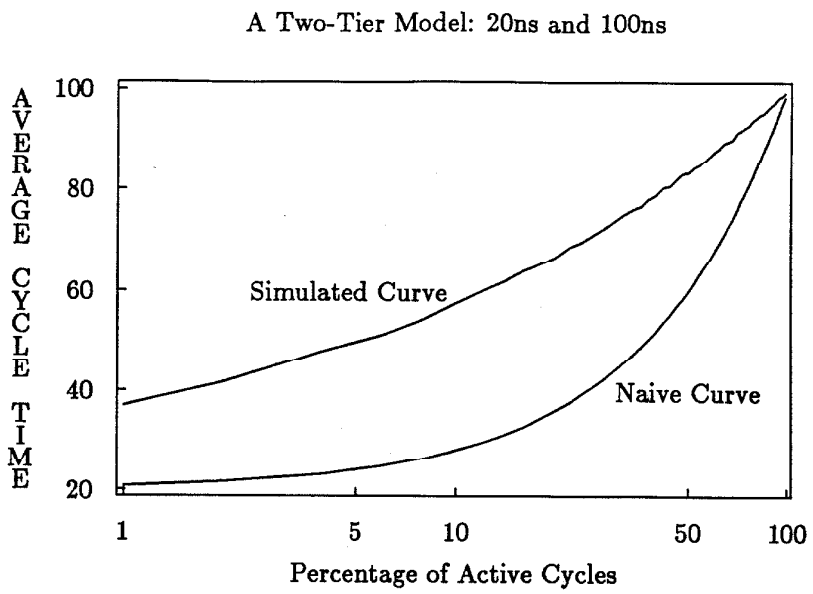

Figure 5.15: Simulated Average Coherent Cycle Delay for a 16 × 16 Mesh

much less silicon area; this may prove to be very important in implementing the adaptive router because of its rather extensive buffer requirements.

For synchronous implementation, these statistical variations must be exploited at a higher organizational level. In particular, the decisions should be organized in a way that can take advantage of the passive cycles where no input- or output-state change occurs. For example, for the assignment logic, it may use these passive cycles to pre-compute a possible future assignment, so that when the actual output channel state change occurs, a simple lookup would suffice to arrive at the decision. The control flow in a scheme like this is necessarily very complicated, and it is likely to require some form of microcoding for implementation. Whether one can devise a ***simple*** synchronous scheme to effectively exploit these statistical variations remains an open question.

5.5 Distributed Clocking

The final topic we shall discuss concerns the system level issue of how to maintain a consistent clock for an arbitrarily large synchronous network. Recall from our feasibility study in Chapter 2 that in order to assure communication deadlock freedom, in spite of the formation of arbitrary loops due to the exploitation of multiple routes, we have chosen to follow the ***no-blocking*** convention. This convention, in turn, requires a strict balance in the input and output data rates across all the communication channels of a node. This required balance can be accomplished by following the coherent channel protocol described in section 2.2.1. The coherent protocol is an example of an ***asynchronous*** handshake protocol that assures secure data transmission between neighbors without making any timing delay assumptions about the respective channels. Perhaps most importantly, the coherent protocol defined in section 2.2.1 results in a network

that is arbitrarily extensible. In particular, the usage of an asynchronous handshake protocol is just a specific example that follows a more general system design discipline: *self-timed* or *delay-insensitive* circuit design [33,48].

The self-timed asynchronous approach is in contrast to the more conventional *synchronous* approach, where information transfer occurs in a lock-step fashion that is coordinated with the use of a system-wide *clock signal*. The synchronous approach has the advantage of being conceptually intuitive; many design and analysis methods for the synchronous model have been thoroughly studied and are widely practiced. However, it suffers from a major drawback in that the distribution of the *global* clock signal across the system becomes progressively more difficult as the size of the computing structures increases, because of the inevitable problem of clock skews and delays [15]. Furthermore, the scaling of VLSI structures is in such a direction that these problems will only get worse as the circuit feature size shrinks [48]. As an alternative to synchronous clocking, in the self-timed approach, circuit elements coordinate their information transfer by following predefined handshake protocols. The self-timed schemes have the advantage that the time required for a communication event between interacting elements need not be determined ahead of time to assure execution correctness. Furthermore, it has the ability to exploit statistical variations in the processing speeds of the individual components in a data-dependent manner. This usually results in an overall system performance that reflects the average rather than the worst-case computation, in contrast to the synchronous clocking approach; this, as we have seen in the previous section, may translate into a possible performance gain.

The advantages obtained in using self-timed asynchronous schemes are not without cost. A drawback of such an approach is that, given the current state of circuit design methodology, self-timed circuits are still fairly difficult in their designs and analyses. This is admittedly a rather subjective opinion. Furthermore, it must be mentioned that recently there has been tremendous progress in understanding the methodologies for designing delay-insensitive VLSI circuits. See [34] for an excellent example. In any case, they can still be rather expensive, in terms of area, to implement. In this section, we explore a third alternative, which represents a *hybrid* of the synchronous clocking and the self-timed handshake approach. In our scheme, on-chip logic is again coordinated by clock signals. However, instead of distributing a single master global clock signal over the entire network, each processing element generates its own *local* clock signal. The elements coordinate their on-chip circuitry according to their locally generated clock signals. Orderly transfer of information across chips is then brought about by synchronizing the independently generated *clock* signals between neighbors. A more restrictive form of this same approach can be found in [54].

5.5.1 The Synchronization Protocol

To proceed, we shall assume as our implementation medium the silicon CMOS technology, and employ the conventional two-phase, nonoverlapping clocking scheme [40]. One basic requirement we shall need is the ability to treat the silicon area covered by a single chip as an *equipotential* region [48]. This requirement is satisfied in the present state of technology with a $2\mu m$ feature size, and shall remain so for even smaller feature sizes as long as long-distance signals are distributed on metal wires.

Our goal is to design a clock-generator circuit to generate the local two-phase

nonoverlapping clock signals $\phi_1, \overline{\phi_1}, \phi_2$ and $\overline{\phi_2}$ on each chip. This circuit will cooperate with its counterparts in its neighboring nodes to enforce the correct synchronization among the corresponding independently generated clock signals. In particular, it enforces locally determined timing constraints in the form of a guaranteed minimum *overlap* between the identical clock phases and a guaranteed *separation* between the opposite clock phases across neighbors. In essence, each node determines from its own considerations, such as local circuit speed and complexities, a minimum clock phase duration that must be satisfied in order for its circuitry to function correctly. The task of the clock generator circuit is to make sure that such requirements are also met by the communicating neighbors. Since each node imposes its own clock speed limit on the network, the clock speed of the tessellation is ultimately determined by the speed of the slowest node, as has been demonstrated in actual simulations.

The correct sequence of clock signal transitions is synchronized by having the neighboring nodes follow a handshake protocol that employs six handshake signals: three from each of the neighboring partners. At the expense of using a more complicated protocol, it is possible to reduce the number of required handshake signals to four. The present protocol is chosen primarily because it is simpler; hence, it can better convey the underlying concepts. The protocol is completely symmetric between the partners, and can be considered to be an extension of the coherent protocol described in section 2.2.1. We represent the local minimum overlap requirement of a node as a *delay* imposed between certain signal transitions. We now define the signals involved. The subscripts i and o are used to distinguish between input and output signals.

- $\phi_1, \overline{\phi_1}$, and $\phi_2, \overline{\phi_2}$ denote the locally generated two-phase nonoverlapping clock signals.
- h_{1o}, h_{2o} and d_o denote the output handshake signals generated by the local clock circuit to its neighbors.
- h_{1i}, h_{2i} and d_i denote the corresponding input handshake signals generated by the clock circuit in its neighbor.
- u denotes the delay initiation signal, and v denotes the delay completion signal. This pair of signals provides an abstraction of the local minimum delay criterion at each node. It can be thought of as input and output of a symmetric delay element whose detail need not concern us at this point.

We now define our protocol in a CSP-like notation [33]:

$$
\begin{aligned}
*[\ & \phi_1\uparrow,\ \overline{\phi_1}\downarrow;\ h_{1o}\uparrow,\ \ [h_{1i}];\ u\uparrow;\ \ [v];\ d_o\uparrow,\ \ [d_i];\ \phi_1\downarrow,\ \overline{\phi_1}\uparrow;\ h_{1o}\downarrow,\ [\neg h_{1i}];\\
& \phi_2\uparrow,\ \overline{\phi_2}\downarrow;\ h_{2o}\downarrow,\ [\neg h_{2i}];\ u\downarrow;\ [\neg v];\ d_o\downarrow,\ [\neg d_i];\ \phi_2\downarrow,\ \overline{\phi_2}\uparrow;\ h_{2o}\uparrow,\ \ [h_{2i}];\ \ ]
\end{aligned}
$$

All three pairs of handshake signals, h_{1o}, h_{1i}; h_{2o}, h_{2i}; and d_o, d_i, interlock with each other to guarantee the *stability* of one another as required in the *firing* rules discussed below. Briefly, the protocol works as follows: The upward transition of h_{1o} and downward transition of h_{2o}, and the *wait* for h_{1i} and $\neg h_{2i}$, are used to initiate and mark the beginning of the minimum delay durations. The two opposite transitions of d_o serve to communicate the satisfaction of the local minimum duration requirement to the communicating neighbors. Symmetrically, the wait for the two opposite transitions of

d_i are used to observe those of the neighbors' requirements. The downward transition of h_{1o} and upward transition of h_{2o}, and the wait for $\neg h_{1i}$ and h_{2i} are used to enforce the separation of opposite clock phases across neighbors.

5.5.2 Circuit Derivation

We now outline informally the reasoning steps that led to our final circuit. We shall be applying the derivation techniques developed for the design of delay-insensitive circuits reported in [33]. First, we observe that the output signals, h_{1o}, h_{2o}, and d_o, always pair up with their corresponding input signals, h_{1i}, h_{2i}, and d_i, in the protocol. Therefore we shall first introduce three new signals to help simplify our reasoning:

$$\begin{aligned} h_{1o} \wedge h_{1i} &\longmapsto h_1\uparrow \\ \neg h_{1o} \wedge \neg h_{1i} &\longmapsto h_1\downarrow \\ h_{2o} \wedge h_{2i} &\longmapsto h_2\uparrow \\ \neg h_{2o} \wedge \neg h_{2i} &\longmapsto h_2\downarrow \\ d_o \wedge d_i &\longmapsto d\uparrow \\ \neg d_o \wedge \neg d_i &\longmapsto d\downarrow \end{aligned}$$

Since these *firing rules* denote the *Muller-C* element behaviors, they can be implemented with three C-elements. Once we have defined these signals, we are ready to list and *strengthen* the whole sequence of signal *firings* for one complete clock cycle. It is remarkable to observe that this sequence represents the only possible sequential firing order as dictated by the causality relationships defined in the protocol. The only possible non-sequential firings are the possible concurrent firings of the clock signals and their inverted forms. Following the notation and methodology specified in [33], we summarize:

$$\neg d \wedge h_2 \longmapsto \phi_1\uparrow, \overline{\phi_1}\downarrow \tag{5.1}$$
$$\phi_1 \wedge \neg\overline{\phi_1} \longmapsto h_{1o}\uparrow \tag{5.2}$$
$$h_{1o} \wedge h_{1i} \longmapsto h_1\uparrow \tag{5.3}$$
$$h_1 \wedge h_2 \longmapsto u\uparrow \overset{\delta}{\longmapsto} v\uparrow \tag{5.4}$$
$$v \longmapsto d_o\uparrow \tag{5.5}$$
$$d_o \wedge d_i \longmapsto d\uparrow \tag{5.6}$$
$$d \vee \neg h_2 \longmapsto \phi_1\downarrow, \overline{\phi_1}\uparrow \tag{5.7}$$
$$\neg\phi_1 \wedge \overline{\phi_1} \longmapsto h_{1o}\downarrow \tag{5.8}$$
$$\neg h_{1o} \wedge \neg h_{1i} \longmapsto h_1\downarrow \tag{5.9}$$
$$d \wedge \neg h_1 \longmapsto \phi_2\uparrow, \overline{\phi_2}\downarrow \tag{5.10}$$
$$\phi_2 \wedge \neg\overline{\phi_2} \longmapsto h_{2o}\downarrow \tag{5.11}$$
$$\neg h_{2o} \wedge \neg h_{2i} \longmapsto h_2\downarrow \tag{5.12}$$
$$\neg h_1 \wedge \neg h_2 \longmapsto u\downarrow \overset{\delta}{\longmapsto} v\downarrow \tag{5.13}$$
$$\neg v \longmapsto d_o\downarrow \tag{5.14}$$
$$\neg d_o \wedge \neg d_i \longmapsto d\downarrow \tag{5.15}$$

$$\neg d \vee h_1 \longmapsto \phi_2\downarrow, \overline{\phi_2}\uparrow \tag{5.16}$$
$$\neg\phi_2 \wedge \overline{\phi_2} \longmapsto h_{2o}\uparrow \tag{5.17}$$
$$h_{2o} \wedge h_{2i} \longmapsto h_2\uparrow \tag{5.18}$$

From the above sequence of firings, we can immediately pair up similar firing rules to get implementations. First of all, we have the following:

$$\begin{aligned}(5) \wedge (14) &\Rightarrow v \,\underline{W}\, d_o \\ (4) \wedge (13) &\Rightarrow (h_1, h_2)\,\underline{C}\, u\end{aligned}$$

where, for example, the handshake output, d_o, is simply the delay completion signal, v, driven off-chip. To generate the clock signals, we observe that instead of generating both the normal and inverted forms independently, one form can be obtained from another simply by using an inverter. This way, we shall have forced explicit sequential dependencies between the clock signals and their inverted forms. By exploiting these dependencies, we can avoid using C-elements to generate the handshake output signals h_{1o} and h_{2o}. In particular, we shall choose to generate the clock signals, $\overline{\phi_1}, \phi_2$ directly, and $\phi_1, \overline{\phi_2}$ by using extra inverters:

$$\begin{aligned}(1) \wedge (7) &\Rightarrow (d, \neg h_2)\,\underline{\vee}\,\overline{\phi_1} \\ (10) \wedge (16) &\Rightarrow (d, \neg h_1)\,\underline{\wedge}\,\phi_2\end{aligned}$$

These, together with the C-elements defined previously:

$$\begin{aligned}(h_{1o}, h_{1i})\,\underline{C}\, h_1 \\ (h_{2o}, h_{2i})\,\underline{C}\, h_2 \\ (d_o, d_i)\,\underline{C}\, d\end{aligned}$$

form our *core* circuit, where we have left out the C-elements required to synchronize signals from multiple neighbors. The complete core circuit is shown in Figure 5.16. Note that we have replaced the signal d by its inverted form, and then pushed the ***bubbles*** related to the inputs of the ***and*** gate and the ***or*** gate forward to become a ***nor*** gate and a ***nand*** gate, which are more natural in CMOS implementation. Observe that because of the symmetry in our handshake protocol, it is possible to connect the output signals, h_{1o}, h_{2o} and d_o, back to the circuit as the input signals, h_{1i}, h_{2i} and d_i. This provides for a systematic way to connect a multiple-neighbor clock-synchronizer module at places such as at the ***edges*** of a mesh, where there are fewer neighbors than required in normal connections.

Some final words on implementation are in order here. First, we observe that while it is difficult to directly implement large delays on MOS, the problem can be nicely circumvented by using an asynchronous binary counter. In fact, implementing the delay using an asynchronous counter actually gives us a ***programmable*** delay element that can be very helpful for testing purposes. Furthermore, if one can obtain an ***a priori*** upperbound on the number of counting transitions needed for the electrical signals to settle, it is possible to hardwire that count on chip and obtain a variable rate clock that scales with such determining physical parameters as feature size and operating temperature. Next, we observe that the signals, $\phi_1, \overline{\phi_1}, \phi_2, \overline{\phi_2}, h_{1o}, h_{2o}$ and d_o, must

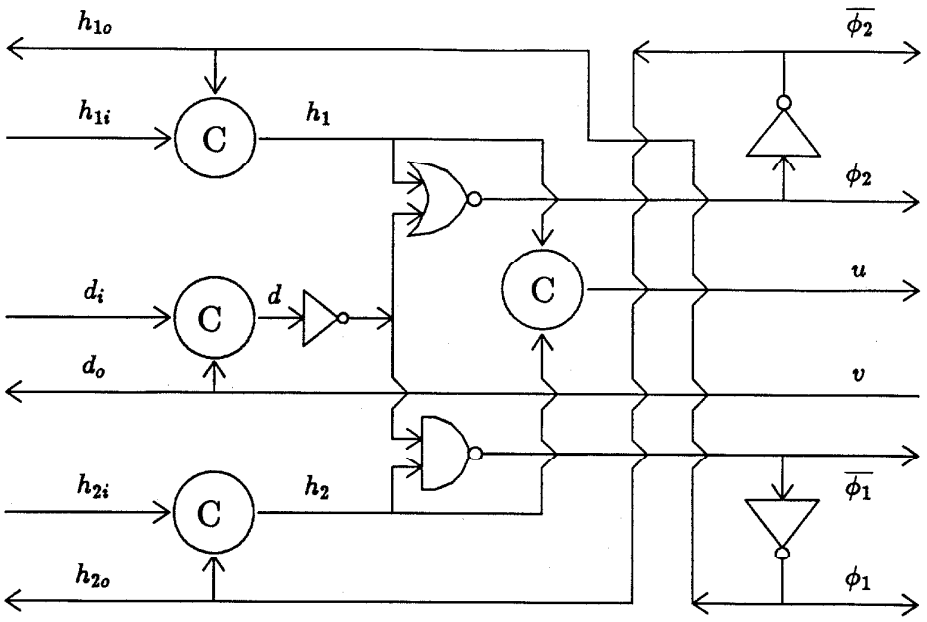

Figure 5.16: Clock Synchronization Circuit

all be either driven off-chip or distributed across the whole chip. Hence each must be buffered by an appropriate driver, and the signals tapped directly at the output of these drivers.

Finally, observe that our circuit as derived has a number of state holding elements. These elements have to be initialized correctly during system *reset* in order for the circuit to operate as desired. We now address this problem. Conceptually, for successful system startup, we want all our nodes to be initialized into the same consistent state within the handshake sequence. If we ignore the handshake actions of h_{1o}, h_{1i}, h_{2o} and h_{2i}, we are left with a simple astable circuit that starts with $\overline{\phi_1}\downarrow$. It continues until it is engaged in the *wait* action $[h_{1i}]$. Our strategy then is to initialize all our nodes to this common state; *ie*, every node in the tessellation engages in the wait action $[h_{1i}]$. This we can do by *forcing* the input, h_{1i}, to low during reset by gating it correctly with the reset signal. In particular, we also force the inputs: h_{2i} to high and d_i and v to low. Meanwhile, during reset, we initialize h_1 to low, h_2 to high, and d and u to low. These initial values constitute the required consistent state of our synchronization circuit. All we need is to have the reset signal be simultaneously applied to all the nodes in the tessellation for a period of time long enough so that all our combinational logic elements have stablized. The removal of the reset signal in each node can now proceed independently and in any arbitrary order, thanks to the speed independence of our circuit.

5.6 Summary

In this chapter, we have examined a number of issues concerning the practical implementation of the adaptive cut-through routing studied in this thesis. Rather than present a final architectural design that is unlikely to emerge until after many design iterations,

we have focused on examining several major realization issues that are either essential to the implementation or are likely to help simplify the final design.

Because of the rather high complexity of any conceivable implementation of the packet priority scheme, we have argued in favor of the much simpler congestion-control protocol described in section 3.5.4 as an approximate technique to assure network progress. Since negative-feedback schemes have been successfully applied in numerous real-life practical systems, and because the simulation results obtained in section 3.5.4 have been very encouraging, we believe that the much simpler protocol provides an excellent practical alternative.

Motivated by the desire to support fast direct cut-through routing, possible header-encoding schemes for both the rectilinear mesh and the octagonal mesh have been examined in detail. A number of encoding formats that provide all relevant routing information in the very first flit have been suggested. Furthermore, these encodings allow the distance information to be incrementally updated with a minimal delay of two clock cycles, which is identical to that used in the oblivious router.

The storage-management issues involved in supporting bounded-length packets have been examined. In particular, it is desirable to decouple the buffer input and output control as much as possible. One solution has been suggested that uses a set of FIFO buffers of a certain capacity to emulate that of a smaller capacity, but that allows very loose coupling between the input and output control.

Based on a conceptual partition of the adaptive control process into a number of simpler tasks, a pipelined control structure has been suggested. The pipeline is organized into four stages in which the odd and even stages are active in alternate half-cycles. This allows the packet data to cut through a node in a minimum delay of two clock cycles. The assignment tasks have been discussed in detail, and a possible iterative logic structure has been suggested. Based on an observation of the statistical percentage of active cycles, the possibility of speeding up the average cycle time using asynchronous handshake logic is suggested and examined.

The issue of maintaining a consistent clock in an arbitrarily extensible synchronous computation structure has been examined. An approach based on synchronizing the clock signals locally across neighbors through the use of a handshake protocol is suggested. The corresponding delay-insensitive asynchronous circuit that enforces the suggested synchronization protocol has been derived.

In addition, a number of open problems related to the above discussions have also been mentioned. Our objective in this chapter is to describe and summarize that which has been learned. This is intended to serve as a starting point for any serious attempt to implement the adaptive cut-through router.

Chapter 6

Conclusions

In this thesis, we have proposed a new framework for performing adaptive multipath routing in multicomputer networks. This framework attempts to exploit the rich connectivity in these networks by allowing the routing control to direct messages along many alternate routes in order to smooth out any temporary congestion and to disperse the local traffic hot spots. The very low transmission-error rates of these networks also allow us to employ cut-through switching; this results in a dramatic reduction in the overall message latency under moderate traffic conditions for messages with reasonable lengths that have to travel across multiple channels.

In order to direct routing in multiple alternate routes, we have adopted a misrouting discipline in our framework that, together with the coherent-channel protocol, allows us to follow a no-blocking convention, yet avoid any buffer overflow. This renders communication deadlock a non-issue. Under this framework, message trajectories are no longer deterministic, but are continuously perturbed by local message loading. Message packets will tend to follow their own shortest-distance routes to destination in normal traffic loading, but can be detoured to other longer but less-loaded routes as local congestion occurs.

Because misrouting destroys the monotonity in message-to-destination distance reduction, and creates the risk of livelock, we have to find separate means to assure the eventual delivery of packets in our framework. Theoretically, we can assign a suitable priority, such as a lexicographic combination of a packet's age and distance from its destination, to each packet, and resolve any channel-access conflicts accordingly. This has been shown to assure delivery of message packets under the consumption assumption.

Another problem we have examined is that of assuring fairness in network access, *ie*, eventual message-packet injection. Fairness in network access is a generic issue in any distributed network, but it is particularly important in our case because of the no-blocking convention, and we have presented two different solutions. The first solution is a passive mechanism that relies on round-trip packets; this is undesirable in that the overhead increases at the precise moment when network bandwidth is insufficient. The second solution is an active feedback mechanism that synchronizes the injection count among neighbors, so that none will fall behind by too much. Because this scheme operates by directly controlling injections, it has been extended to provide congestion control.

In order to obtain a first-hand understanding of the dynamics governing the different

network parameters and their relationships to the overall performance of the network, we have studied the performance behaviors both analytically, through stochastic modeling, and also experimentally, through extensive traffic simulations. Based on a bisection bandwidth argument, we have obtained a theoretical upper bound on the average steady-state injection rate per node for the general class of k-ary-n-cubes and meshes under random message traffic. The result has a general form that is inversely proportional to the radix of the network. This agrees with our intuition that because the network bisection bandwidth grows at a sublinear rate, the amount of traffic each node can produce will decrease as the network size increases. Similarly, the sum of the average packet length and average message distance forms a theoretical lower bound on the average packet latency. These bounds provide a uniform frame of reference for the interpretation of performance results obtained from simulations.

All of our simulation experiments indicate that regardless of the connection topology and routing scheme used, the network performance behaviors can always be partitioned into two regions: First, there is an unsaturated region in which the network throughput closely follows the applied load, the average message latency stays very close to the theoretical lower bound, and the average source queueing time is practically zero. As the applied load increases, the network shifts into a saturation region in which the network throughput stays at a stable constant saturation value, but the average message latency increases rapidly, and the average source queueing time becomes unbounded. A difference between the adaptive and the oblivious schemes is the applied load at which this transition takes place. Simulation results indicate that the adaptive scheme enjoys approximately a factor of two increase in the applied load before the network becomes saturated.

These network behaviors are different from most conventional communication networks where the throughput and other performance metrics will actually deteriorate at very heavy applied loads. It appears that the main reason for this difference is that in conventional networks packets are *discarded* when a receiving node's buffer is full. Under very heavy traffic, where the buffers at every node are likely to be filled to capacity, this leads to a disastrous drop in the number of successful packet transmissions, and results in a reduced overall network throughput.

The congestion-control protocol, which is an extension of the network-access fairness-assurance protocol, has proven to be very effective in confining the network operating points to the unsaturated region that delivers acceptable network performance. Because the protocol directly monitors the occurrence of packet misrouting at each node, it has the effect of minimizing the amount of misrouting. This results in a general improvement in network behavior around the transition region, and provides a practical alternative to assuring packet delivery in our adaptive scheme.

We have explored the possibility of improving the reliability of large-scale multicomputer networks by performing fault-tolerant routing under the adaptive framework. In particular, our investigation focused on finding an effective scheme that is both simple and practical, and that is subject to the constraints present in multicomputer networks. Our approach relies on the ability of the adaptive scheme to exploit the inherent path redundancies that are already provided by the richly connected topologies popular in multicomputer networks. Rather than attempting to devise a fault-tolerant routing algorithm, which appears to be difficult because of the restriction of using only local

information, and which is likely to be topology dependent, we instead introduced the notions of a convex subset and a communication kernel that characterize the conditions under which we can continue to use the original algorithmic routing relations to systematically direct routing in faulty networks. Such a strategy also satisfies our desire to reasonably maintain the high performance achieved in networks that directly implement the routing operation in hardware, since it allows us to continue to use, with minimal change, the original routing hardware for the non-faulty networks.

The actual regularization procedure proceeds both by selectively discarding certain nodes that are difficult to reach and by selectively restraining certain nodes to operate as pure switches. As one would have expected, the simulation and computation results indicate that our strategy is more effective for a network that is more richly connected, *eg*, the binary-n-cube, than for a network with a connectivity that is comparatively sparse, *eg*, the 2D rectilinear mesh. As an example of how to improve the regularization yield for 2D structures, we suggest the 2D octagonal-mesh network, which is formed by augmenting the rectilinear mesh with additional diagonal channels. This new 2D network admits excellent fault-tolerant capabilities under our adaptive-routing framework. Both the performance and the reliability characteristics of the octagonal mesh have been studied, and the preliminary results indicate that it is a serious candidate that deserves further in-depth investigation.

As a practical implementation alternative to the use of the complicated priority-based delivery-assurance scheme, we suggest the use of the very simple negative-feedback-based congestion-control protocol described in section 3.5.4 as an approximate technique for assuring network progress. The protocol can be implemented with one extra wire per channel and a few simple gates at each node. Another implementation issue that we examined is how to encode the header of a packet for both the rectilinear mesh and the octagonal mesh, using a relatively narrow flit width. We have suggested the use of more complex alphabets to encode the required delta displacement values in a sign-and-magnitude format. The signs of these delta values provide all relevant routing information in the very first flit of the header. Furthermore, these alphabets encode additional information that allows the encoded number to be incrementally updated with a minimum delay of two clock cycles; this is identical to that used in the oblivious router.

Based on the conceptual partition of the adaptive control process into four simpler tasks, *ie*, buffer allocation, profitable packet-to-channel assignment, misrouting assignment, and header modification, we suggest a four-stage pipelined control structure in which the odd and even stages are active in alternate half-cycles. Using such a pipelined control allows the packet data to cut through a node with the desired minimum delay of two clock cycles. The same conceptual partition of the adaptive control also suggests a corresponding datapath for the adaptive router that employs an input crossbar to switch the input packets into the internal buffers, and an output crossbar to switch the output packets from the internal buffers. The corresponding buffer- and channel-assignment tasks can be readily implemented by using suitable iterative logic structures. Since we are primarily interested in networks that are either very large or are intended to be arbitrarily extensible, we have developed an asynchronous handshake protocol between nodes that allows us to maintain a set of consistent clocks in a network of synchronous nodes. The circuitry within each node is coordinated by the locally generated clock

signals, and the protocol synchronizes these local clock signals between neighbors. The synchronization protocol is enforced by a very simple delay-insensitive asynchronous circuit.

In summary, we have studied the possibility of performing adaptive multipath routing in order to improve the performance and reliability of multicomputer networks. To this end, we have proposed a new framework for performing adaptive cut-through routing in these networks, and we have organized our study around four major area of interests: establishing the theoretical feasibility, investigating the performance behaviors, exploring the potential reliability enhancements, and examining the major implementation issues of our proposed adaptive-routing framework. Our study has been generally successful and the positive results give us hope that the adaptive scheme can indeed be made to improve on the already highly evolved oblivious scheme.

In addition to those studied in this thesis, there are a number of other issues and open problems that clearly require further study. For example, one area of interest that deserves further investigation concerns how to actively employ misrouting in order to get around traffic hot spots. In all our performance investigations, we have always adopted a conservative policy toward misrouting, using it only to defensively avoid buffer overflow. The main reason is that it is very difficult to perform discretionary misrouting based only on local information, and it is unlikely that misrouting of any form will help in a network with heavily loaded traffic. In fact, from the congestion-control traffic experiments, we have actually observed improvements in network performance by minimizing misrouting under such heavy traffic conditions. However, it is still conceivable that some form of *randomized* misrouting [60] may be helpful in situations where the network is not globally congested. The challenge will be to devise a scheme for reacting quickly and for keeping the amount of misrouting under control when the network traffic conditions change.

Another area of interest that deserves further investigation concerns how to effectively perform broadcasting under our adaptive scheme. There are a number of issues that require attention here. In broadcasting, it is generally necessary to duplicate a received packet in order to transmit it across a multiple number of channels. Such an action may lead to buffer overflow if it is not handled carefully. Additional problems can arise because of the misrouting requirement. For instance, a packet received may not be transmitted across all of its desired output channels, but instead be sent to other previously transmitted channels. Closely tied to this is the issue of how to determine the termination of the broadcasting action in face of this nondeterminism.

Yet another area of interest that naturally suggests itself for further investigation is that of fault-tolerance, where quite a number of issues remain to be resolved. First and foremost is the problem of fault diagnosis, *ie*, how to determine if a channel or a neighbor has malfunctioned. For example: If a neighbor is dead and therefore unable to participate in the coherent protocol, how long should a node wait before it times out? Furthermore, in our discussion of fault-tolerant routing in Chapter 4, we have adopted the idealized notion of a fail-stop processor, *ie*, one that will stop completely rather than send spurious messages when a fault occurs. In practice, this condition may not be satisfied, and a scheme must be installed to guard against such dangers. A second problem that demands investigation is the study of how to carry out the regularization procedure on-the-fly as faults are incrementally discovered. Closely tied to this is a whole set of higher-level abstraction issues, such as end-to-end error correction and the

retransmission of missing packets, that have to be imposed on top of our basic scheme in order to support truly fault-tolerant computation.

Another related problem that requires study concerns the scaling behavior of our regularization approach. For example, it is doubtful that the impressive regularization yield displayed by the 2D octagonal mesh can be maintained as we increase the size of the mesh. In fact, a deterioration in yield is likely to be observed for any network with a bisection bandwidth that grows at a sublinear rate. Given any *fixed* nonzero fault probability, as the size increases, such a network will most likely be disconnected into many small disjoint components. The question is to determine how large the size of a network can increase, while still being able to deliver favorable yield statistics, before the scaling effect takes over. To this end, the octagonal mesh enjoys a favorably large yield margin, *eg*, $\approx 85\%$ yield at 10% faults even at 1024 nodes. This observation reinforces our earlier conclusion that the octagonal mesh is a serious candidate for further investigation.

Yet a third problem related to fault-tolerance concerns the finding of alternate 3D structures that will deliver favorable yield statistics. Three-dimensional structures, like the two-dimensional counterparts, have their own attractive properties. For example, in addition to being physically realizable, the extra degree of freedom typically creates many more paths between nonadjacent nodes. This is an important advantage for fault-tolerance in light of the scaling issue discussed above. Furthermore, for a machine of size N, the bisection bandwidth capacity of a 3D structure scales at a rate that is $O(N^{\frac{2}{3}})$, which is much faster than the $O(N^{\frac{1}{2}})$ rate for a corresponding 2D structure. This disparity generally dictates that the larger machines be configured into 3D structures. Similarly, for machines of the same size, the average message distance will generally be much shorter in a 3D configuration than in a 2D configuration. Some of the 3D lattice structures, such as the *body-centered cubic* lattice and the *face-centered cubic* lattice, that have been studied by chemists appear to be promising candidates for further investigation.

While there are clearly many additional challenging areas of interest to pursue, perhaps the most challenging of all will be to realize, on silicon, the adaptive routing control studied in this thesis.

Bibliography

[1] G.A. Agha, *Actors: A Model of Concurrent Computation in Distributed Systems*, MIT Press, 1986.

[2] W.C. Athas and C.L. Seitz., "Multicomputers: Message-Passing Concurrent Computers," *IEEE Computer*, August 1988, pp. 9–24.

[3] W.C. Athas, *Fine-Grain Concurrent Computation*, TR:5242:87, Computer Science Department, Caltech.

[4] B. Becker and H. Simon, "How Robust is the n-Cube?" *Proc. 27th Ann. IEEE Symp. Foundations Comput. Sci.*, Oct. 1986, pp. 283–291.

[5] V.E. Benes, *Mathematical Theory of Connecting Networks and Telephone Traffic*, Academic Press, New York, 1965.

[6] D. Bertsekas and R. Gallager, *Data Networks*, Prentice Hall Inc., Englewood Cliffs, N. J. 07632, 1987.

[7] N.J. Boden, *A Study of Fine-Grain Programming Using Cantor*, Caltech Technical Report CS-TR-88-11, 1988.

[8] A. Borodin and J. Hopcroft, "Routing, Merging, and Sorting on Parallel Models of Computation," *Journal of Computer and System Sciences*, Vol. 30, 1985, pp. 130–145.

[9] R.E. Blahut, *Theory and Practice of Error Control Codes*, Addison-Wesley, Mass. 1983.

[10] K.M. Chandy and J. Misra, "Distributed Computation on Graphs: Shortest Path Algorithms," *CACM*, Vol. 25, No. 11, Nov. 1982, pp. 160–177.

[11] W.J. Dally and C.L. Seitz, "The Torus Routing Chip," *Distributed Computing*, 1986(1), pp. 187–196.

[12] W.J. Dally and C.L. Seitz, "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks," *IEEE Transactions on Computers*, Vol. C-36, No. 5, May 1987, pp. 547–553.

[13] W.J. Dally, *A VLSI Architecture for Concurrent Data Structures*, Kluwer Academic Publishers, 1987.

[14] C.M. Flaig, *VLSI Mesh Routing Systems*, Caltech Computer Science Department Technical Report, 5241:TR:87.

[15] A.L. Fisher and H.T. Kung, "Synchronizing Large VLSI Processor Arrays," *IEEE Transactions on Computers*, Vol. C-34, No. 8, August 1985, pp. 734–740.

[16] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, 1979.

[17] M. Gerla and L. Kleinrock, "Flow Control: A Comparative Survey," *IEEE Transactions on Communications*, Vol. COM-28, No. 4, April 1980, pp. 553–574.

[18] J.M. Gordon and Q.F. Stout, "Hypercube Message Routing in the Presence of Faults," *The Third Conference on Hypercube Concurrent Computers and Applications*, Vol. 1, ACM Press, 1988, pp. 33–36.

[19] K.D. Gunther, "Prevention of Deadlocks in Packet-Switched Data Transport Systems," *IEEE Transactions on Communications*, Vol. COM-29, No. 4, April 1981, pp. 512–524.

[20] J. Hastad, T. Leighton, M. Newman, "Reconfiguring a Hypercube in the Presence of Faults." *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, May, 1987.

[21] F.C. Hennie, *Finite-State Models For Logical Machines*, John Wiley and Sons, Inc., 1968.

[22] T.C. Hu, *Combinatorial Algorithms*, Addison-Wesley, 1982.

[23] Intel Scientific Computers, *iPSC User's Guide*, Order No. 175455-001, 15201 N. W. Greenbrier Parkway, Beaverton, Oregon, August 1985.

[24] Intel Scientific Computers, *iPSC/2 Brochures and Application Software Material*, Order No. 280110-001, 15201 N. W. Greenbrier Parkway, Beaverton, Oregon.

[25] P. Kermani and L. Kleinrock, "Virtual Cut-Through: A New Computer Communication Switching Technique," *Computer Networks* Vol. 3, No. 4, Sept. 1979, pp. 267–286.

[26] L. Kleinrock, *Communication Nets: Stochastic Message Flows and Delay*, McGraw-Hill, New York, 1964.

[27] L. Kleinrock, *Queueing Systems, Vol. 1 & 2*, John Wiley & Sons, New York, 1975, 1976.

[28] L. Kleinrock and F. Kamoun, "Hierarchical Routing for Large Networks: Performance Evaluation and Optimization," *Computer Networks*, Vol. 1, 1977, pp. 155-174.

[29] C.R. Lang, Jr., *The Extension of Object-Oriented Languages to a Homogeneous, Concurrent Architecture*, TR:5014:82, Computer Science Department, Caltech.

[30] D.C. Little, "A Proof for the Queueing Formula: $L = \lambda W$," *Operations Research*, Vol. 9, May 1961, pp. 383–387.

[31] M.T. Liu, "Distributed Loop Computer Networks," *Advances in Computers*, M. Yovits, Academic Press, 1978, pp. 163–221.

[32] C. Lutz, *et. al.*, "Design of the Mosaic Element," *Proceedings of the MIT Conference on Advanced Research in VLSI*, Artech Books, 1984, pp. 1–10.

[33] A.J. Martin, "A Synthesis Method for Self-timed VLSI Circuits," *Proc. 1987 IEEE International Conference on Computer Design: VLSI in Computers & Processors*, IEEE Comp. Soc. Press, 1987, pp. 224–229.

[34] A.J. Martin, *et. al*, "The Design of an Asynchronous Microprocessor," *Advanced Research in VLSI: Proceedings of the 1989 Decennial Caltech Conference on VLSI*, MIT Press, 1989, pp. 351–373.

[35] R.J. McEliece, *Finite Fields for Computer Scientists and Engineers*, Kluwer Academic Publishers, 1987.

[36] R.J. McEliece, *The Theory of Information and Coding*, Addison-Wesley, Mass., 1977.

[37] P. Merlin, and P. Schweitzer, "Deadlock Avoidance in Store-and-Forward Networks – I : Store-and Forward Deadlock," *IEEE Transactions on Communications*, Vol. COM-28, No. 3, March 1980, pp. 345–354.

[38] Motorola Inc. "MC88100 Technical Summary: 32-Bit Third-Generation RISC Microprocessor," Semiconductor Technical Data, 1988.

[39] J.M. McQuillan, I. Richer, and E. C. Rosen, "The New Routing Algorithm for the ARPANET," *IEEE Transactions on Communications*, Vol. COM-28, 1980, pp. 711–719.

[40] C. Mead and L. Conway, Chapter 3, *Introduction to VLSI Systems*, Addison-Wesley, 1980.

[41] J.Y. Ngai, *A Framework for Adaptive Routing*, TR:5246:87, Computer Science Department, Caltech.

[42] J.Y. Ngai and C.L. Seitz, "A Framework for Adaptive Routing in Multicomputer Networks," *Proceedings of the 1989 ACM Symposium on Parallel Algorithms and Architectures*, Santa Fe, NM.

[43] C.H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Prentice Hall, Englewood Cliffs, N.J. 1982.

[44] D.K. Pradhan, Chapter 7, *Fault-Tolerant Computing: Theory and Techniques*, Vol. 2, Prentice Hall, Englewood Cliffs, N.J. 1986.

[45] W. Reisig, *Petri Nets: An Introduction*, Springer-Verlag, 1985.

[46] R.D. Schlichting and F.B. Schneider, "Fail-Safe Processors: An Approach to Designing Fault-Tolerant Computing Systems," *ACM Transactions on Computer Systems*, Vol. 1, No. 3, August 1983, pp. 222–238.

[47] C.L. Seitz, "The Cosmic Cube," *CACM*, Vol. 28, No. 1, January 1985, pp. 22–33.

[48] C.L. Seitz, "System Timing," *Introduction to VLSI Systems*, C. Mead & L. Conway, Addison-Wesley, 1980, Chapter 7.

[49] C.L. Seitz, J. Seizović, and W-K. Su, "The C Programmer's Abbreviated Guide to Multicomputer Programming," TR:88-1, Computer Science Department, Caltech.

[50] C.L. Seitz, "Concurrent VLSI Architectures," *IEEE Transactions on Computers*, Vol. 33, No. 12, Dec. 1984, pp. 1247–1265.

[51] C.L. Seitz, *et. al.*, "The Architecture and Programming of the Ametek Series 2010 Multicomputer," *The Third Conference on Hypercube Concurrent Computers and Applications*, Vol. 1, ACM Press, 1988, pp. 33–36.

[52] J. Seizović, *The Reactive Kernel*, Caltech Technical Report CS-TR-88-10, 1988.

[53] C.S. Steele, *Placement of Communicating Processes on Multiprocessor Networks*, TR:5184:85, Computer Science Department, Caltech.

[54] W-K. Su, *Supermesh*, TR:5125:84, Computer Science Department, Caltech.

[55] W-K. Su, *Ph.D. Thesis*, To be published, Computer Science Department, Caltech.

[56] A.S. Tanenbaum, *Computer Networks*, Prentice Hall, Englewood Cliffs, N.J., 1981.

[57] H.C. Tijms, *Stochastic Modelling and Analysis: A Computational Approach*, John Wiley & Sons, 1986.

[58] C.D. Thompson, *Complexity Theory for VLSI*, Technical Report CMU-CS-80-140, Department of Computer Science, Carnegie-Mellon University, August 1980.

[59] S. Toueg and J.D. Ullman, "Deadlock-Free Packet Switching Networks," *Proc. ACM Symposium on the Theory of Computing*, Atlanta, Georgia, May 1979, pp. 89–98.

[60] L.G. Valiant, "A Scheme for Fast Parallel Communication," *SIAM J. Computing*, Vol. 11, No. 2, 1982, pp. 350–361.